



吴 迪 编著

零基础学Qt 4编程

目錄

零基础学 qt4 编程

作者：wd007

来源：《零基础学Qt4编程》专栏

第 1 章 走近 Qt

本章重点

- 了解 Qt 的起源和功用
- 了解 Qt 产品线以及对应的平台
- 了解 Qt 开发工具的组成和功用
- 掌握 Qt 的授权区分、服务支持
- 掌握 Qt4.5 版的最新内容

1.1 Qt 简介

首先让我们看看业界对于 Qt 的评论：

“Qt 将帮助我们为用户提供空前‘诱人’的应用程序。诺基亚对跨平台 Qt 库和 Linux 平台的一贯投入，是免费软件桌面和移动设备堆栈创新的主要驱动力。 ”

Mark Shuttleworth， Ubuntu 项目创始人

“Qt 是极具创新的和实用的技术。包含了 QtWebKit 的 Qt 提供了一个强大的，跨平 台的网络体验，确保了我们的 3D 环境所具备的身临其境的感觉。我们渴望将 QtWebkit 纳 入到‘Second Life‘中，从而提高在 Web 上的虚拟世界的集成度。”

Linden LabJoe Miller,

Linden 实验室平台与技术开发部副总裁 看起来，Qt 是如此的吸引人，那么就让我带您走进 Qt 的世界吧。

1.关于 Qt

Qt 是一个已经形成事实上的标准 的 C++ 框架，它被用于高性能的跨平台软件开发。 除了拥有扩展的 C++类库以外，Qt 还提供了许多可用来直接快速编写应用程序的工具。此 外，Qt 还具有跨平台能力并能提供国际化支持，这一切确保了 Qt 应用程序的市场应用范 围极为广泛。

自 1995 年以来，Qt 逐步进入商业领域，它已经成为全世界范围内数千种成功的应用 程序的基础。Qt C++ 框架一直是商业应用程序的核心。无论是跨国公司和大型组织（例 如：Adobe®、Boeing®、Google®、IBM®、Motorola®、NASA、Skype®）、还是无数小型公司和组织都在使用 Qt。Qt 也是流行的 Linux 桌面环境 KDE 的基础（KDE 是所有主要的 Linux 发行版的一个标准组件）。Qt4 在新增更多强大功能的同时，旨在比先前的 Qt 版本更易于 扩展和使用。Qt 的类功能全面，提供一致性接口，更易于学习使用，可减轻开发人员的工 作负担、提高编程人员的效率。另外， Qt 一直都是完全面向对象的，并且允许真正的组件编程。

Qt 软件前身为 Trolltech （奇趣科技），2008 年 6 月被诺基亚收购。更多关于 Qt Software 的信息，请访问网站 www.qtsoftware.com。

下面是 Qt4 的 Logo，Qt 通常以清新的绿色作为自己的宣传色，图中中间区域是一个大 写的 Q 字母，里面斜向嵌入了大写的 T 字母，在右上角表明了 Qt 的版本。



2.关于 Trolltech （奇趣科技）

图 1-1 Qt4 的 Logo

Trolltech 是由 Haavard Nord (执行总裁) 和 Eirik Chambe-Eng (总裁)于 1994 年创立的，2008 年 6 月被 NOKIA 收购。过去十几年来，Trolltech 的销售业绩持续增长。 Trolltech 采取了成功的双重授权战略，为开发者提供商业和免费软件的授权使用。 它的创始人秉持财富共享的理念，已捐献出 Trolltech 公司 5%的资产给慈善基金会。 它拥有遍布全球 60 个国家的 4400 个客户，其中包括 Adobe， IBM（国际商用机器公司），Sharp（夏普）、Siemens（西门子）以及其他全球知名公司。目前，拥有来自 17 个不同国家的 雇员。Trolltech 公司的总部位于挪威的奥斯陆，并在澳大利亚的布里斯班，中国北京及 美国加利福尼亚的帕洛阿尔托设有办事处。

3.关于 Nokia(诺基亚)

诺基亚是移动世界的全球领先者，引领着互联网及通信融合产业的转型与发展。凭借多样化的移动终端产品、软件与服务，诺基亚为人们带来了音乐、导航、视频、电 视、影像、游戏、移动商务等丰富体验。面向用户的互联网服务的发展以及企业解决方案和软件业 务的增长是诺基亚发展的重点。此外，诺基亚还通过诺基亚西门子通信公司为通信网络提供设备、解决方案和服务。

4. Qt 支持的平台（以 4.5 版为准）

Qt4.5 可提供于下列平台:

- Windows (Microsoft Windows Vista,XP, 2000, 2003,NT4, Me/98)
- WinCE
- Mac (Mac OS X)
- X11 (Linux, Solaris, HP-UX, IRIX, AIX, ... 以及其他 UNIX 系统)
- Embedded Linux

表 1-1 所示为 Qt4.5 支持的平台和编译器的详细情况，表 1-2 为 Qt4.5 不支持的平台 和编译器的详细情况。

表 1-1 Qt4.5 支持的平台情况

软件平台	硬件架构	**Makespec**	编译器
Microsoft Windows	Intel 32/64-bit	win32-g++, win32-icc, win32-msvc2003, win32-msvc2005 win32-msvc2008	GCC 3.4.2 (MinGW) (32-bit), Intel icc, MSVC 2003, MSVC 2005 (32 and 64-bit), MSVC 2008
Windows CE	Intel 32-bit, ARMv4i, MIPS	Wince-msvc2005, wince-msvc2008	Visual Studio 2005 Visual Studio 2008
Linux (32 and 64-	Intel 32/64-bit,	Linux-g++	GCC 3.3,
bit)	Itanium,	linux-icc	GCC 3.4,
MIPS	linux-icc-32	GCC 4.0,	
linux-icc-64	GCC 4.1, 4.2, 4.3		
Embedded Linux	ARM,	qws/linux-arm-g++,	GCC 3.4,
Intel 32-bit,	qws/linux-x86-g++,	GCC 4.1,	
MIPS,	qws/linux-g++	GCC 4.2,	
PowerPC	GCC 4.3		
Apple Mac OS X (32-bit)	Intel 32/64-bit, PowerPC	Macx-g++ macx-g++42	GCC 4.0.1, GCC 4.2
Solaris	SPARC, Intel 32-bit	Solaris-cc solaris-g++	Sun CC 5.5 GCC 3.4.2
AIX	PowerPC	Aix-xlc aix-xlc-64	xlC 6
HPUX	PA/RISC,Itanium	Hpux-acc hpux-g++ hpux-g++-64 hpuxi-acc	A.03.57 (aCC 3.57) GCC 3.4.4 A.06.10 (aCC 6.10)

表 1-2 Qt4.5 不支持的平台情况

软件平台	硬件架构	**Makespec**	编译器
Windows XP/Vista	Intel 32/64-bit	win32-msvc	Visual C++ 6.0
Windows XP/Vista	Intel 32/64-bit	win32-msvc2002	Visual Studio 2002
Windows XP/Vista	Intel 32/64-bit	win32-msvc.net	Visual Studio 2002
IRIX	MIPS	irix-cc	MIPS Pro
IRIX	MIPS	irix-g++	GCC 3.3

1.2 Qt 纪事概览

Qt Software 前身为 Trolltech（奇趣科技），Trolltech 始创于 1994 年

- 1996 年 Qt 上市
- Qt 已成为数以万计的商业和开源应用程序的基础
- Qt 的软件授权机制具有经受市场检验的双重授权（开源与商业）模式
- Trolltech 于 2008 年 6 月被 Nokia 收购，加速了其跨平台开发战略 阅读材料：Qt 简史

Qt 工具包最初是在 1995 年 5 月变为公众可用的。它最初由 Haavard Nord(Trolltech 的 CEO) 和 Eirik Chambe-Eng(Trolltech 的总裁)开发。Haavard 和 Eirik 是在位于挪威特 隆赫姆的挪威科技学院相识的，在那里他们都获得了计算机科学硕士学位。

Havvard 对 C++图形用户界面开发的兴趣始于 1988 年，当时他被一家瑞典公司委托设 计并且开发一个 C++图形用户界面工具包。

几年后，在 1990 年夏天，Havvard 和 Eirik 因为一个超声波图像方面的 C++数据库应 用程序在一起工作。这个系统要求能够在 Unix、Macintosh 和 Windows 上都能运行。那个 夏天中的一天，Haavard 和 Eirik 出去散步享受阳光。当他们坐在公园的一个长椅上， Haavard 说："我们需要一个面向对象的显示系统。"由此引发的讨论奠定了他们即将创建 的面向对象的多平台图形用户界面工具包的智力基础。

Haavard 于 1991 年开始和 Eirik 合作设计、编写最终成为 Qt 的这些类。随后的一年， Eirik 提出了"信号和槽"的设想——一个简单并且有效的强大的图形用户界面编程范例。 Haavard 实践了这个想法，并且建立了一个手写代码实现。到 1993 年，Haavard 和 Eirik 当时已经开发出了 Qt 的第一个图形核心并且能够实现他们自己的窗口部件。同年末， Haavard 提议他们一起开展并且创建"世界上最好的 C++图形用户界面工具包"的业务。

1994 年成为两个程序员不幸的一年，他们没有客户，没有资金，只有一个未完成的产 品，却希望闯入一个稳定的市场。非常幸运的是他们的妻子都有工作并且愿意支持他们的丈 夫。在这两年里，Haavard 和 Eirik 认为需要继续开发他们的产品并且开始盈利。

他们选择"Q"作为类的前缀，是因为该字母在 Haavard 的 Emacs 字体中看起来非常漂亮。"t"代表"工具包(toolkit)"，是从"Xt"，X 工具包中获得的灵感。公司于 1994 年 3 月 4 日创立，最初名为"Quasar Technologies"，后更名为"Troll Tech"，现在改 为"Trolltech"。

1995 年 5 月，通过 Haavard 大学时的一个教授的联系，挪威 Metis 公司与他们签订了 一份基于 Qt 进行软件开发的合同。大约同一时间，Trolltech 雇佣了 Arnt Gulbrandsen， 正是他设计并实现了一套有独创性的文档系统，并且对 Qt 的代码也作出了贡献。

1995 年 5 月 20 日，Qt 0.90 被上传到 sunsite.unc.edu。6 天后，在 comp.os.linux-announce 上发布。这是 Qt 的第一次公开发布。Qt 可以被同时用于 Windows 和 Unix 开发，

它在两种平台上提供了相同的应用程序编程接口。从第一天起， Qt 就提供两种许可版本：

一种是进行商业开发所需的商业许可版本，另一种是进行开源开发的自由软件版本。 Metis 的合同确保了 Trolltech 的发展，在 10 个月之内没有人购买 Qt 的商业许可。

1996 年，European Space Agency 购买了 10 份 Qt 商业许可，成了第 2 个 Qt 客户。凭 着坚定的信念，Eirik 和 Haavard 又雇佣了另外一个开发人员。Qt 0.97 在 5 月底发布，并 且在 1999 年 9 月 24 日，Qt1.0 面世。在这一年底，Qt 已经发展到了 1.1 的版本，共有来 自 8 个不同国家的客户，他们购买了 18 份商业许可。这一年，在 Matthias Ettrich 的带 领下，创立了 KDE 项目。

Qt 1.2 在 1997 年 4 月发布。Matthias Ettrich 利用 Qt 建立 KDE 的决定似的 Qt 成为 在 Linux 环境下开发 C++图形用户界面的实际标准。Qt1.3 在 1997 年 9 月发布。

Matthias 在 1998 年加入 Trolltech。当年 9 月最后一个 1 系列的版本 1.40 发布。 1999 年 6 月 Qt2.0 发布。Qt2.0 有很多架构上的改变，比它的前几个版本功能更为强大， 更为成熟。它还具备了 40 个新类和 Unicode 支持。Qt2 有了一个新的开源许可，Q 公共许 可(QPL，Q Public License)，它遵循了开源定义。在 1999 年 8 月，Qt 赢得了 LinuxWorld 的最佳库/工具奖。大约在这个时候，Trolltech Pty Ltd（澳大利亚）建立了。

Trolltech 在 2000 年发布了 Qt/Embedded。它被设计为可以运行在嵌入式 Linux 设备 上，并且提供了它自己的窗口系统作为 X11 的轻型替代品。现在 Qt/Embedded 和 Qt/X11 除 了提供商业许可之外，还可以提供被广泛使用的 GNU 通用许可（GPL，GNU General Public License）。到 2000 年底，Trolltech 已经创建了 Trolltech Inc.（美国），并发布了第 一个 Qtopia 版本，一个手持设备环境。Qt/Embedded 在 2001 年和 2002 年两次获得 LinuxWorld“Best Embedded Linux Solution”（最好的嵌入式 Linux 解决方案）奖。

Qt3.0 在 2001 年发布。现在 Qt 可以用于 Windows、Unix、Linux、嵌入式 Linux 和 Mac OS X。Qt3.提供了 42 个新类和超过 50 万行的代码。Qt3.0 在 2002 年赢得了 Software Development Tools 的“Jolt Productivity Aword（震撼生产力奖）”。

2005 年夏，Qt4.0 发布，它大约有 500 个类和 9000 多个函数，Qt4 比以往的任何一个 版本都要全面和丰富，并且它已经裂变成多个函数库，从而使开发人员可以根据自己的需要 只连接所需要的 Qt 部分。相对于以前的所有 Qt 版本，Qt4 的进步是巨大的，它不仅彻底的 对高效易用的模板容器、高级的模型 /视图功能、快速而灵活的二维绘图框架和强大的统一 字符编码标准的文本查看和编辑类进行了大量改进，就更不必说对那些贯穿整个 Qt 类中的 成千上万个小的改良了。现如今， Qt4 具有如此广泛的特性，以至于 Qt 已经超越了作为图 形用户界面工具包的界限，逐渐成长为一个成熟的应用程序开发框架。 Qt4 也是第一个能够 在其所有可支持的平台上既可用于商业开发又可用于开源开发的 Qt 版本。

同样在 2005 年，奇趣公司在北京开设了一家办事处，以便为中国及其销售区域内的用 户提供服务和培训，并且为 Qt/Embedded Linux 和 Qtopia 提供技术支持。

通过获取一些非官方的语言绑定件（Langauge binding），非 C++程序员也已早就开始 使用 Qt，特别是用于 Python 程序员的 PyQt 语言绑定件。2007 年，公司发布了用于 C#程序员的非官方语言绑定件 Qyoto。同年，Qt Jambi 投放市场，它是一个官方支持的 Java 版 Qt应用

程序编程接口。

自奇趣公司（现已被 NOKIA 收购）诞生以来，Qt 的声望经久不衰，而且至今仍然保持 高涨。取得这样的成绩不但说明了 Qt 的质量，而且也说明了人们都喜欢使用它。在过去的 10 年中，Qt 已经从一个只被少数专业人士所熟悉的“秘密”产品，发展到了如今遍及全世 界拥有数以千计的客户和数以万计的开源开发人员的产品。

1.3 Qt 套件的组成（以 Qt4.5 为准）

自 4.5 版开始，Qt 首次以 SDK 形式发布了 Qt 套件，并在单独的安装程序中包含了完整 的 Qt SDK。

Qt SDK 在一个单独安装程序内包含了使用 Qt 进行跨平台开发所需的全部工具，其中 包括：

1.Qt Creator - 跨平台 IDE

Qt Creator 是全新的跨平台集成开发环境 (IDE)， 专为 Qt 开发人员的需求量身定 制。它包括：

- 高级 C++ 代码编辑器
- 集成的 GUI 外观和版式设计器-Qt
- 项目和生成管理工具
- 集成的上下文相关的帮助系统
- 图形化调试器（基于 GDB）

从这些话语中，我们不难看出 Nokia 全力打造 Qt Creator 的决心，意图将以前单独列 出的 Qt Designer、Qt Assistant、Qt Linguist 全部整合到 Qt Creator 中，把它们全部 作为 Qt Creator 的一部分，从而奠定 Qt Creator 的“官方出品、根正苗红”的地位。关 于 Nokia 的战略想法这里暂且不谈，虽然 Nokia 意图如此，但是笔者觉得还是有必要罗嗦 两句，向读者朋友介绍一下 Qt SDK 中的几个核心成员：

2.Qt 库

Qt Library

是一个拥有超过 400 C++类，同时不断扩展的库。它封装了用于端到端应用程序开发 所需要的所有基础结构。优秀的 Qt 应用程序接口包括成熟的对象模型，内容丰富的集合类，图形有户界面编程与布局设计功能，数据库编程，网络， XML，国际化，OpenGL 集成等等。

Qt Designer

是一个功能强大的 GUI 布局与窗体构造器，能够在所有支持平台上，以本地化的视图 外观与认知，快速开发高性能的用户界面。

Qt Assistant

是一个完全可自定义，重新分配的帮助文件或文档浏览器，又称作 Qt 助手。它的功能

类似于 MSDN，支持 html 的子集（图片、超链、文本着色）， 支持目录结构、关键字索引和全文搜索， 可以很方便的查找 Qt 的 API 帮助文档 ，它是编程人员必备、使用频率最高的工具之一。

Qt Demo

是 Qt 例子和演示程序的加载器，有了这个工具， 用户可以很方便的查看 Qt 提供的多 姿多彩的例子程序，从中不仅可以看到程序运行的情况，还可以查看源码和文档。

qmake

是一个用于生成 Makefile（编译的规则和命令行）的命令行工具。 它是 Qt 跨平台编 译系统的基础。 它的主要特点是可以读取 Qt 本身的配置， 为程序生成平台相关的 Makefile。

uic

是一个用来编译 ui 文件的命令行工具，全称是 UI Compiler。 它能把.ui 文件转化 为编译器可以识别的标准 C++文件， 生成的文件是一个.h。 这个工具通常情况下不需要用 户去手动调用， qmake 会帮你管理.ui 文件和调用 uic 工具。

moc

是一个 用来生成一些与信号和槽相关的底层代码的预编译工具 。全称是 Meta Object Compiler， 即元对象编译器。该工具处理带有 Q_OBJECT 宏的头文件， 生成形如 moc_xxx.h, moc_xxx.cpp 的 C++代码，之后再与程序的代码一同编译。 同样，这个命令行工具也不需要用户手动调用， qmake 会在适当的时候调用这个工具。

rcc

是一个 Qt 的资源文件编译工具。 Qt 的资源系统是自己一套特别的设计， 工程中可以 包含后缀为 qrc 的资源文件， 由 rcc 工具根据.qrc 文件中的内容将相关的文件编译为二进 制，并与源码编译在一起，保存在应用程序的二进制文件中。 这个命令行工具同样不需要 手动调用，一般由 qmake 调配使用。

qtconfig

是一个在 X11 系统下用于配置 Qt 环境的工具。 它可以设定 Qt 环境的字体、Style、Palette、打印机等。 它的设定信息会保存在用户的 home 目录下，所以可以按不同用户的 喜好来设定不同的值。

3. 翻译和国际化工具

Qt Linguist

是一套用来消除国际化工作流程中所带来障碍的工具，又称作 Qt 语言家。 开发小组 可把应用程序的翻译转换外包给非技术性翻译人员，从而可增加精确度，大大加快本地化处 理过程。

lupdate

是 Qt 国际化的重要命令行工具之一， 它的功能是从源码文件或其他资源文件中提取 需要翻译的字符串，并将之用正确的编码和格式存入 ts 文件中。 这个 ts 文件是 xml 格式 的普通文本文件， 但不建议用普通的文本编辑工具来编辑， 最好的方法是用 Linguist 来 处理这个文

件。

lrelease

是 Qt 国际化的重要命令行工具之一， 它负责将 ts 文件转化为程序使用的 qm 文件。 转化过程最大的变化是去掉了原始文件中所有的空白和未翻译的内容， 并将存储格式压缩， 所以 qm 文件是保留所有有效信息但占用硬盘最少的格式。

1.4 Qt 的授权

Qt 产品的提供是采用双重授权的软件许可模式。在该双重授权模式下， Qt 产品不仅可 在获得商业许可下针对专利软件开发，而且还可以在 GPL（通用公共许可证，版本 2 或版本 3）下用于开发免费和开源软件。 也就是说，Qt 这个软件本身是开源和免费使用的， 如果 你基于 GPL 协议来开发软件的话，你开发的东西都要以 GPL 协议发布一 开源并免费提供源 码。

自从 Qt4.5 版本发布以后，Qt 通过三种授权方式提供： 商业、 LGPL 和 GPL，并且 仍然符合 Qt 的双重授权战略，表 1-3 示出了 Qt 的授权方式的简要说明，表 1-4 对比了授 权方式在具体使用时的不同。

表 1－3 Qt 的授权方式

Qt 商业版	**Qt** 商业授权适用于开发专属和/或商业软件。此版本适用于不希望与他人共享源代码，或者遵循 **GNU** 宽通用公共许可证 **(LGPL) 2.1** 版或 **GNU GPL 3.0** 版条款的开发人员。
Qt GNU LGPL v. 2.1	此版本 Qt 适用于开发专属或开源 Qt 应用程序，前提条件是必须遵循 GNU LGPL 2.1 版的条款。
Qt GNU GPL v. 3.0	如果您希望将 Qt 应用程序与受 GNU 通用公共许可证 (GPL) 3.0 版本条款限制的软件一同使用， 或 者您希望 Qt 应用程序遵循该 GNU 许可证版本的条款，则此版本 Qt 适用于开发此类 Qt 应用程 序。

表 1－4 Qt 授权对照表

	商业版	**LGPL** 授权版	**GPL** 授权版
授权收费	收取授权费	免费	免费
必须提供更改 Qt 的源代码	不需要, 更改的代码可以不公 布	必须提供源代码	必须提供源代码
可以创建专属应用程序	可以， 不必公布源代码	可以， 但必须遵循 LGPL v.2.1 条款	不可以， 应用程序受 GPL 限 制， 且源代码必须公开。
提供更新	是， 仅限申请有效维护服务的 用户。	是， 免费发布。	是， 免费发布。
支持	提供， 条件是维护协议必须有 效。	不提供， 必须单独购买。	不提供， 必须单独购买。
运行时收费	是	否	否

按照授权协议的不同，Qt 被按不同的版本发行：

Qt 商业版用于商业软件的开发，提供免费升级和技术支持服务。

Qt 开源版是 Qt 的非商业版本，是为开发自由和开放源码软件提供的 Unix/X11 版本。 在 GNU、GPL 或 LGPL 许可证下，它可以免费下载和使用。

此外，Qt 还提供了免费评估版、快照、beta 测试版、预览版等多种版本，其中免费评 估版 Qt 适用于 Windows、Mac、Linux、嵌入式 Linux 和 Windows CE 平台，它不但具备 全部功能，还带有源代码，Nokia 会在您进行评估期间提供技术支持。而快照、 beta 测试 版、预览版等版本则得不到 Qt 的支持。

1.4.1 Qt 开源版和商业版的不同

在网上经常看到有朋友提问，Qt 开源版和商业版到底有那些不同，有没有必要使用商 业版等问题，看来还是很有必要向大家介绍一下：

1. 功能不尽相同

两者在源码上基本一致，但开源版缺少一些数据库插件，因为这些插件都是基于特定数据库客户端程序的，很多商业数据库的客户端程序并不是开源的，所以插件就无法开源；也就是说，开源版不支持商业数据库的驱动，一般需要大家自己写驱动或者是采用第三方的 驱动。另外，在 Windows 版本上，开源版没有 ActiveQt 这个模块，它可以用来开发ActiveX 程序。

2.收费不同

开源版不收费，商业版根据版本不同，费用不同，一般一个 developer license 需要 大约几千美金。

3.服务不同

开源版不能享受服务，但可以到一些开放的 maillist 和论坛讨论；商业版有一年的免 费技术支持，有问题就直接发给 support@qtsoftware.com， 另外商业版中还包括一年的同 产品免费下载支持。

4.协议不同

这个应该是最本质的不同，使用开源版开发需要遵循 GPL 或者 QPL，而使用商业版就没 有这个限制，大家可以看看 license agreement，原则上只要不开发和 Qt 竞争的产品就可以了。

总而言之，商业版 Qt 授权包括电子邮件(email)支持，可以获得升級，让您能够开发 完全闭源的软件。LGPL 对用户重新连接代码库的许可权有一些限制，不能提供商业数据库 如 Oracle 等的驱动，以及对 Micorsoft Office 二次开发的支持等，有时还有强加某些机 构可能不喜欢的架构要求等其他限制。除此之外，大多数情况下开源版的功能与商业版并无 二致。商业版与开源版功能的详细比较如表 1－5 所示：

表 1—5 开源版与商业版的比较

功能模块	开源版	商业版	
Qt 的基本模块（工具、核心、窗口部件、对话框） 与平台无关的 Qt 图形用户界面工具包和应用类	√	√	
Qt 设计器 可视化的 Qt 图形用户界面的生成器	√	√	
图标视图模块 几套图形用户交互操作的可视化效果	√	√	
工作区模块 多文档界面（MDI）支持	√	√	
OpenGL 三维图形模块 在 Qt 中集成了 OpenGL	√	√	
网络模块 一些套接字，TCP、FTP 和异步 DNS 查询并且与平台无关的类	√	√	
画布模块 为可视化效果，图表和其它而优化的二维图形领域	√	√	
表格模块 灵活的可编辑的表格/电子表格	√	√	
XML 模块 通过 SAX 接口和 DOM 的很好的且已经成形的 XML 解析器	√	√	
SQL 模块 SQL 数据库访问类	部分驱动 (Sqlite、MySQL)	√	
ActiveQt 模块	支持 Office 二次开发等的模块	x	√
售后服务和支持	x 可单独购买服务	√	

如果要下载上述版本的 Qt，只需访问网址： http://www.qtsoftware.com/downloads，而商业版本还需要与 Nokia 接洽。

1.5 Qt 的产品

Qt

Qt 是一个完整的 C++应用程序开发框架。它包含一个类库和 一系列用于跨平台开发及 国际化的工具。Qt API 在所有支持的平台上都是相同的，Qt 工具在这些平台上的使用方式也一致， 因而 Qt 应用的开发和部署与平台无关。

Qtopia

Qtopia 是一个面向嵌入式 Linux 的全方位应用程序开发平台，同时也是用于基于 Linux 的 PDA（个人数字助理），智能电话（Smartphone）以及其他移动设备的用户界面。

Qt/Embedded

Qt/Embedded 是一个完整的自包含 GUI 和基于 Linux 的嵌入式平台开发工具。

1.6 Qt 的服务与支持

1. Qt 标准支持 — 12 个月

与很多人预想的有所不同，实际上，获得任何 Qt 产品的商业、LGPL 或 GPL 授权的 客户都可以获得 Qt 标准支持。这是很有吸引力的一项人性化的市场策略，也让广大用户对 Qt 产品增强信心，因为这项举措实际上表明了：Qt 不同版本的产品都具有一致的高品质和 优质服务。

标准支持主要面向的是应用程序开发人员。 Qt 在以下几个方面为客户提供一般性的建 议和指导：Qt API、功能、方法和编程技巧，乃至提供示例，还包括：产品使用、安装和技 术支持，在线程序缺陷跟踪， 产品升级提示等。

每个购买了 Qt 商业授权的客户都会获得一年的标准支持和维护服务，并且可以每年 在有效期到期前重新申请购买下一年度的支持和维护服务。Qt 标准支持还可通过使用 Qt 开源授权（例如 LGPL）方式作为单独的产品进行购买。

举个例子，如果你购买了 Qt 标准支持，那么通常会在 4 个工作日内收到针对你个人的书面解答，答复直接由进行实际操作的技术专家提供，这些专家里甚至可能包括 Qt 软件开发团队中的成员。

2. Qt 优先支持 — 在一个工作日内回复

Qt 优先支持是在标准支持之上的升级，这需要你付出额外的费用，获得的好处是可以 在服务有效年限内通过专门的高优先级支持渠道获得 10 次优先支持，Qt 会为您指定专门 的支持工程师，他会在一个工作日内为您优先解答困惑和问题，值与不值，你可以自己衡量 一下再作出决定。

3. Qt 的版本支持 — x.y.z 规则

Qt 的支持服务仅适用于官方发布的版本，不适用于快照、 beta 测试版、预览版和其他 不支持的版本，另外前面提到的 Qt 免费评估版则在评估期间可以享受到 Qt Software 的支 持。

Qt 对产品版本 x.y.z 提供为期 1 年的支持和维护，直到后续版本（ x.[y+1.0.0].z 或 [x+1.0.0].y.z，以先发布的为准）发布日的一年后为止。举例来说， Qt 3.3.2 版将在 Qt3.4.2 或者是 Qt4.3.2 版发布满 1 年后终止支持和维护，由于 Qt 3.4.2 先于 Qt 4.3.2 发 布，因此以 Qt3.4.2 版的发布时间为准，这就是 Qt 版本支持维护的 x.y.z 规则，听起来有 些“绕”，但当你登录到 http://www.qtsoftware.com/downloads，面对所有罗列到一起的 Qt 版本时，你就会明白了。

1.7 Qt 的最新进展

1.7.1 增加协议

诺基亚宣布，从 Qt 4.5 版本起，其用于桌面和嵌入式平台的 Qt 跨平台用户界面(UI) 及应用程序框架将在开源 LGPL 2.1 版授权下提供。此前，Qt 一直是在通用公共授权（GPL）下提供给开源社区的。另外，现已可以通过新的网站名称 www.qtsoftware.com 来了解 Qt 。

向 LGPL 的转移将为开源和商业开发人员提供比 GPL 更多的授权许可证，从而为开发 人员提高了灵活性。此外，Qt 源代码库將更加开放，鼓励更多來自桌面和嵌入式系统的开发人员社区的贡献。随着这些变化，开发人员将能够积极推动 Qt 框架的演进。

Qt 4.5 同时也可在商业授权条款下使用，Qt 之前版本的授权则保持不变，也就是说， 比如你要使用 Qt4.4.3 开源版的话，就需要遵守 GPL 而不是 LGPL 协议。而且，Qt 的 服务将扩展，以确保所有 Qt 开发专案，无论选择何种授权，都能获得同等支持。这一举措的效果如何呢，请看下面的来自业界的回应：

“Qt 在 LGPL 条款下的使用，让运用基于 Qt 应用程序顶端的 KDE 组件创建应用程 序的授权合理化。“ KDE e.V. 董事会成员 Sebastian Kügler 说，“这一更多权限的授 权为 Qt 和 KDE 技术的推广再次降低了门槛。KDE 团队欢迎开放开发进程，并期待以此进 一步促进 KDE 和 Qt Software 部门的协作。"

“结合诺基亚独立于操作系统的应用程序框架 Qt 和飞思卡尔的可实施软件，为 OEM和应用程序开发人员在挑选飞思卡尔芯片时提供了特有的自由度，从而允许开发人员为其应用程序开发和维护单一的代码库。“飞思卡尔解决方案及可实施技术副总 裁 Raja Tabet 说，“LGPL 模式是一个出色的和时效性的授权选择，这将加速结合了飞思卡尔和 Qt 的平台的推广与开发。”

“Qt 被广泛应用于 Kubuntu 和 KDE 应用程序中，Canonical 很高兴看到其在授权模 式上的这一突破，” Ubuntu 项目创始人 Mark Shuttleworth 说，“Qt 新的授权条款将帮 助我们为用户提供空前‘诱人’的应用程序。诺基亚对跨平台 Qt 库和 Linux 平台的一贯 投入，是免费软件桌面和移动设备堆栈创新的主要动力。 ”

“我们欢迎诺基亚简化 Qt 授权的举措”，Linden 实验室平台与技术开发部副总裁 Joe Miller 说，“我们发现 Qt 是耐人寻味且极具创新的技术，无论授权方式如何，这个新的授权方式已经使得我们在追求将 QtWebkit 集成到 Second Life 时所作的决策变得更为简单。”

从中我们可以看出，诺基亚的这一举措是有力的，得到了来自世界各地用户的普遍欢 迎。尤为重要的是，它的推出为 Qt 进一步的大规模商业应用扫清了障碍。

如果想要了解更多的发布信息，可以访问：http://www.qtsoftware.com/about-us- cn/licensing 。

1.7.2 开放源代码库

诺基亚宣布，从 2009 年 5 月起，Qt 源代码库面向公众开放，Qt 开发人员可通过为 Qt 以及与 Qt 相关的项目贡献代码、翻译、示例以及其他内容，协助引导和塑造 Qt 未来的发 展。为了便于这些内容的管理，Qt Software 启用了基于 Git 和 Gitorious 开源项目的 Web 源代码管理系统，网址为：http://qt.gitorious.org 。

在推出开放式 Qt 代码库的同时，Qt Software 在 qtsoftware.com 发布了其产品规划（Roadmap)。其中概述了研发项目中的最新功能，展现了现阶段对 Qt 未来发展方向的观点，以期鼓励社区提供反馈和贡献代码，共创 Qt 的未来。

该消息以及关于贡献模式和产品规划的详细内容现在发布在官方中文网站。 想了解 更多， 请参考以下链接：

全部的消息内容：http://www.qtsoftware.com/about-us-cn/news/qt-contribution- model-announced

贡献模式：http://www.qtsoftware.com/resources-cn/the-qt-contribution-model 产品规划：http://www.qtsoftware.com/resources-cn/qt-roadmap

1.7.3 确定 Qt 的发展方向

诺基亚 QtSoftware 首席技术官 Benoit Schillings 称 Qt4.5 为应用程序开发树立了标 杆。他还表示：“通过 Qt 性能方面的改进、QtCreator 的诞生和 Qt 软件开发工具包的发 布，那些寻找能够将本地内容和 Web 开发完整集成的应用框架的开发人员将如虎添翼，凭 增开发活力与灵活表现。”

从 Nokia 发布的 Qt Software RodeMap 路线图上，我们可以看到，在最新的 Qt4.5 以

及后续版本中，Qt 将在以下方向持续改进：

1.不断增强 Qt 的跨平台能力 主要体现在：

- 支持 Mac OS X Cocoa 框架
- 增加了对 Windows CE 上 Phonon 和 WebKit 模块的支持
- 推出 Qt for S60 (Tech Preview)
- 增加 Windows 7 支持
- 推出 32 位/64 位版本，更新 Qt 以便适用于 64 位 Mac。 2. 持续提升 Qt 的性能

主要体现在：

- 提高了图形绘图性能
- 改进 Web 和混合式开发，支持动态的集成 web 和本地内容 3.不断完善 Qt 工具包

主要体现在：

- 推出 Qt Creator 并不断增强其功能
- 继续提供并改进 Eclipse 和 Visual Studio 插件
- 研究并准备推出社区和协作工具
- 支持混合式应用程序开发
- 构建高效 Build 系统 4. 支持先进的界面开发

主要体现在：

- 推出 Qt Kinetic 项目
- 增加手势触摸功能支持
- 提供 OpenVG 支持功能
- 增加 3D 支持工具

5. 展开 Qt 框架发展方向研究 主要体现在：

- 多媒体服务
- 混合式应用程序开发
- 内存和资源管理
- XML Schema (模式) 支持
- Qt 3D 可移植
- 下一代对象视图

看了上面的介绍，广大的 Qt 用户和潜在的使用者一定会极大的增强信心，毫无疑问的 是，被 NOKIA 收购后，Qt 将获得更好更快的发展，并且会一如既往的坚持开源与商业的双 重授权策略。

1.7.4 Qt 4.5—Qt 发展的重要里程碑

Qt4.5 的发布，可以看作是 Qt 发展史上的又一个重要的里程碑。借此， Qt 第一次提出 了“Qt EveryWhere”的口号，真正的全速前进走向了大规模的商业应用，开源社区也得到 了更加强有力的支持，尤为突出的是 Qt 的性能得到了很大的提升。用一句话来总结，就是 Qt 从未充满如此奔放的活力和富有如此强大的号召力。

由于 Qt 4.5 版是自 Trolltech 被 NOKIA 收购后，Qt 发布的首个全新版本，因此显得格 外引人注目，在国内外掀起了学习的热潮。 Qt 4.5 变化较大，在很多方面与以前的 Qt4 系 列有所不同，为了使初学者不致于走弯路，下面就对 Qt4.5 的变化给大家做一个详细的介 绍。

1. 修改授权方式，增加 LGPL 协议 支持－Qt 更开放

Qt4.5 增加了对 LGPL 协议的支持，并继续 支持多种授权协议如 GPL、GNU 等，这为 Qt 和 KDE 的商业应用进一步扫清了障碍。

2.增加 SDK 包，可以直接安装－Qt 更易用

首次增加了 Qt SDK 包，提供了直接安装的版本，在一个单独安装程序内包含了使用

Qt 进行跨平台开发所需的全部工具，其中包含了 Qt 库、Qt Creator IDE、Qt 开发工具， 并且仍然像以前那样提供二进制的 tar 包。

3. 大幅度提升性能－Qt 更便捷

性能提升是 Qt 4.5 的主要设计目标之一。通过以重构关键功能、采用全新的插件式 图形系统、推出全新的性能基准库－ QtBenchLib 等举措，基于 Qt 的应用程序的运行时间 性能得到了大幅提高。

更为详细的性能提升比较，请参见 Qt Labs 上的度量标准。

4. 在 X11 上的测试平台是 Kubuntu

目前在 x11 上，Qt4.5 已在 Kubuntu8.04、8.10 上验证测试过了，但 Qt Software 并未提及其它的发行版。所以如果你想在 Linux 上使用 Qt4.5 的话，Kubuntu 将是一个不错 的选择。

5. 区分为 32/64 位版

在 4.5 版以前，Qt 是不区分 32 位和 64 位版的，这次明确区分了，所以大家在使用时 需要注意，你的软硬件平台是否对应支持，不要 "张冠李戴"了。

6.正式支持 WinCE 这个不用多说了，以前发布的版本都是测试版。

7.支持 Symbian 上的 S60 平台

截至 09 年 5 月，Qt 发布了 Qt for S60 预先发布版 "Garden" ，正式的版本将在 09 年中发布。

8.不完全支持 MIPS

大家如果拿到了 Qt4.5 的 SDK，可以注意一下，其名字中均带有 x86 字样，经过官方 证实，目前 Qt4.5 在 MIPS（比如国产 CPU 龙芯）上还不能直接使用，要使用的话，需要采 用编译的方式安装，并且要修改某些文件的内容，步骤比较繁杂，难度较大。不过已经有网 友在龙芯上编译 Qt Creator 成功，详情可以参见 Qt 知识库网站。（见附录网址）

9.在 Mac 上支持 64 位 Cocoa

Qt 4.5 在全新的 Mac Cocoa API 基础上对 64 位应用程序开发提供支持，这样 Qt 开 发人员就可将需要大量资源的应用程序部署到最新版本的 Mac OS (10.5) 上。

10.升级至 WebKit 最新版本

在 Qt 4.5 中，Qt WebKit Integration 现已使用 WebKit 最新版本，其中包括：

- 支持 Netscape 插件 (NPAPI)，可在您的 Qt 应用程序中加入 Flash™ 内容
- 支持 HTML 5，包括缩放、基于 CSS 的动画以及更多功能
- 集成精简的 SquirrelFish JavaScript 引擎 11. 附加许多新功能

Qt 4.5 引入了许多其他跨框架的新功能，其中包括：

- 用于 QtScript ECMA 标准 Qt 脚本引擎的全新调试器 (观看视频!)
- 支持 XSLT，可将 XML 内容转换为 XML、HTML 或其他文本
- 支持开放文档格式 (.odf)
- 改进了代理支持

12. Qt Jambi 将交由社区开发

Qt 与 Java 编程语言绑定的最终新功能版 Qt Jambi 4.5.0_01 已经发布，根据 NOKIA 发表的声明，Qt Jambi 的开发将在一年的维护期结束后停止，以便集中资源进行 Qt 跨平 台应用和 UI 框架的开发。为了确保 Qt Jambi 实施的连续性，该项目会在

http://qt.gitorious.org 上保留，社区开发者仍可继续对其进行开发。

通过这次发布，Nokia 确定了 Qt Software 的发展方向一就是在保持 Qt 跨平台发展这 个主线的同时，能够适应更多“有前景”的软硬件平台，比如 Nokia 的 Symbian 操作系统 上的 S60 等，并提供对开发全生命周期提供完整的支持。

总之一句话，Qt 采用了诸多新的举措，努力使得 Qt 比以往更开放、更易用以及更便捷。

有关 Qt 4.5 包含的新功能的详细信息，请查阅 Qt 技术文档。

在不同的平台上，Qt 提供的功能是不尽相同的，表 1-6 所示为 Qt4.5 在所支持的平台 上的功能分布，表 1-7 所示为 Qt4.5 的类库的核心功能，大家可作为参考。

表 1-6 Qt4.5 在各个平台上的功能分布

功能模块	Linux/X11	Embedded Linux	Windows	Windows CE	Mac OS X
拖放可视化 GUI 构建器	√	√	√	√	√
国际化和 翻译工具	√	√	√	√	√
可定制的 HTML 帮助文件阅读器	√	√	√	√	√
集成 Eclipse 和 KDevelop IDE	√	x	x	x	x
集成 Visual Studio	x	x	√	√	x
一整套可定制的 UI 控件或 widget	√	√	√	√	√
本地 Aqua 外观， 以及 Aqua 风格的 widget	x	x	x	x	√
集成了 OpenGL，支持 3D 图形	√	√	√	√（OpenGL ES）	√
集成了 Direct3D® 的 3D 图形支持	x	x	√	x	x
强大的 多线程功能	√	x	√	x	√
可处理上百万个对象的 2D 图形画布	√	x	√	x	√
集成了 Phonon 多媒体框架	√	√	√	x	√
WebKit 集成	√	√	√	x	√
网络、XML 和数据库功能	√	√	√	√	√
ECMA 标准脚本引擎	√	√	√	√	√
紧凑高效的视窗系统 (QWS)	x	√	x	x	x
虚拟帧缓冲 (QVFb)	x	√	x	x	x

表 1-7 Qt4.5 类库的核心功能

功能模块	详细内容
先进的图形用户界面（GUI）	Qt 使用所支持平台的本地化图形 API，充分利用系统资源并给予应用程序本地化的界面。
基于 OpenGL® 与 OpenGL® ES 的 3D 图形	虽然 OpenGL 完美支持 3D 图形，但却不支持创建应用程序用户界面。Qt 通过与 OpenGL 的紧密集成解决了这一难题。并且支持 Windows 平台上的 Direct3D®
多线程	Qt 的跨平台多线程功能简化了并行编程， 另外它附加的同步功能可以更 加轻松地利用多核架构。
嵌入式设备的紧凑视窗系统	可以把基于 Qt 的应用程序直接写入 Linux 帧缓冲，解除了开发者对 X11 视窗系统的需求。
对象间通讯	提供信号与槽机制，使应用程序能够在不同的组件间通信
2D 图形	Qt 给您提供一个功能强大的 2D 图形画布，用以管理和集成大量的图形元素。
多媒体框架	Qt 使用 Phonon 多媒体框架为众多的多媒体格式提供跨桌面与嵌入式操作系统的回放功能。Phonon 可以轻松将音频与视频回放功能加入到 Qt 应用程序当中，并且在每个目标平台上提取多媒体格式与框架。
WebKit 集成	Qt WebKit 集成，即 Qt 集成了 WebKit 功能，WebKit 是 KDE 项目下基于 KHTML 的开放源 web 浏览器引擎。目前 Apple®，Google™ 与 Nokia 等公 司使用 Qt WebKit 集成。
网络连接	Qt 让您网络编程更简单，并支持跨平台网络编程
XML	Qt 为 XML 文件以及 SAX 和 DOM 协议的 C++实现，提供了一个流媒体文件读写器。同时 Qt 还包含了 XQuery – 一个简单的类似 SQL 的查询语言，用于解析 XML 文件来选择和聚合所需要的 XML 元素，并且将它们转换 成 XML 输出或其它格式的输出。
脚本引擎	Qt 包含一个完全集成 ECMA 标准的脚本引擎。 QtScript 提供 QObject 集成，把 Qt 的信号与槽机制整合成脚本，并且实现了 C++ 与脚本的集 成。
数据库	Qt 帮助您将数据库与您的 Qt 应用程序无缝集成，支持所有主要的数据库驱动，并可以多种视图或数据识别表单方式显示数据

1.8为什么选择 Qt

前面说了这么多 Qt 的介绍，相信一定有很多朋友还是要问，那么我们为什么要选择 Qt 做开发呢，它到底有哪些突出的优势呢，下面就是笔者给出的答案。

- Qt 是基于 C++ 的一种语言扩展(Extention)

  C/C++ 目前还是一种很多人都在学习的语言. Qt 的好处就在于 Qt 本身可以被称作是一种 C++ 的延伸。Qt 的类都是用 C++ 写出来的。这也就是说, Qt 本身已经继承了 C++的快 速、简易、面向对象等许多的优点。

- Qt 具有非常好的可移植性（Portable）

  Qt 不只是可以在 Linux 中运行，也同样可以运行在 Microsoft Windows 、Mac OS X 等多种不同的平台中。这也意味者 ,利用 Qt 编写出来的应用程序,在几乎不用修改的情况下,就可以同时在多种平台中运行。 Qt 的应用非常之广泛，从 Linux 到 Windows ，从 x86 到 Embedded 等都有大量 Qt 应用的成功范例。

- Qt 支持跨平台构建（Cross-Platform Builds）

  编写用于多平台的软件是单调乏味的，且可能随时出现错误。维护编制文件更是如此，尤其是当不同的编译器和平台组合需要若干个编制文件时。通过 qmake 工具，Qt 能够很好地面对这一挑战。这个工具可以为目标平台生成准确无误的编制文件。

- Qt 是开源软件(Open Source)

  Qt 产品提供的是采用双重授权的软件许可模式。在该双重授权模式下， Qt 产品不仅可在获得商业许可下针对专利软件开发，而且还可以在 GPL（通用公共许可证，版本 2 或版本 3）下用于开发免费和开源软件。

- Qt 架构健壮，性能强大(powerful)

  Qt 已由成千上万商业与开放源应用程序员，在多个操作系统与编译器上进行了测试， 奠定了高性能应用程序的基础。此外，Qt 运行时无需依赖“虚拟器”，模拟层或大容量的运行时间环境。它如本地化的应用程序一样，能够直接写入低级的图形函数，因而使用 Qt 开发的应用程序能以源代码速度执行。

- Qt 使用起来简便高效（concise）

  Qt 统一的跨平台 API 让程序员们集中精力致力于可增值的技术革新，而无须担心维护 和管理现有应用程序多版本的基础结构与界面。 因此 Qt 开发人员仅需要学会一种 API 来写入应用程序，该程序便可在任何地方运行。

Qt Software 投入了相当大的努力使得 Qt 使用起来尽可能简单和直观。来自世界各地 的客户反馈普遍认为 Qt 编程简单而有趣；而对于商务应用而言， Qt 可以转换为更多的功 能，并且在保持质量性能的情况下，只需更少的维护工作。

总之，Qt 可以跨平台、不依赖虚拟机机制，速度和性能不会比同类型的 Java, .NET(C#)差，这就是程序员选择 Qt 的最直接的理由了。

阅读材料

GTK+, Qt, wxWidgets 比较简评

这位来自台湾的朋友把 GTK+, Qt, wxWidgets 比较分析得相当到位。很明显的，MFC 注 定将淡出江湖了。下面就是几个跨平台库的横向比较。

之前因为把 MFC 痛骂了一顿，有网友在询问其它 GUI toolkit 的相关事项，所以小弟 分享一下个人对三大知名图形界面库的简短评论，以下纯属个人主观意见，不是专业人士， 所知有限，望前辈不吝指正或补充。

1.先讲 GTK+

GTK+ 主要用在 X Window 上，整个设计的架构和许多概念和 MFC 以及一般 Windows 上的程式开发大异其趣，入门门槛较高，而且最主要的特色是，它用不具有物件功能的纯 "C" 語言，模擬物件导向。 所以写起来比较复杂艰涩，而且充满大量巨集，使用和除错都 不是很容易，但优点则是可以用 C，不需 C++，如果和 Win32 SDK 比較，不会难学多少， 缺点是不易上手使用，而且文件比较缺，架构又非常复杂，且提供的东西比起其它无所不包 的 library，是简陋了一点，函数命名又臭又长。对于简单的程式， GTK+会显得太复杂，但 是当你开始想扩充其它 library 也都沒提供的进阶功能，就会开始赞叹 GTK+ 的架构严 谨，还有超乎想象的高度弹性。同样的东西要用 MFC 來做反而会要人命，並且对多国語言 的支援良好，內部也全面使用 UTF-8，相容性好，又是 unicode。能夠习惯的话，GTK+ 值 得推荐，但沒有很建议学，毕竟不好学，要用到熟会需要比較久，而且那样很多 C++ 的功 能会用不到。 GTK+ 有 C++ 版本叫做 GTK--，沒用过，但看文件觉得，并沒有比 gtk+ 简 单到那里去。 因为 gtk+ 本來就是物件导向，所以即使換了 c++ 語言，写起來架构还是差 不多的。 另外，gtk+ 有 Windows 版本，但缺点是，执行缓慢，不稳定，而且界面是使用 gtk+ 自己的，不是使用 Windows 內建的"Native" 原生图形界面，看起来会不太习惯。

Mac OS X 下可用 X11 來執行 gtk+，但那样出來的程式是长得像 UNIX 程式，而不是美美 的 OS X Aqua 外观。

2.再说 wxWidgets

wxWidgets 和 MFC 最接近，命名习惯或架构都高度相似，会 MFC 的话几乎不用重新学 习。它有十余年历史，此外，它的物件封裝比 MFC 要好，提供的功能也多上太多，又跨平 台。一般知名的 MFC 程式都会选择用 wxWidgets 改写，來快速移植原程式到其他平台。例 如，eMule 用 wxWidgets 移植出 aMule, xMule, 还在开发中的 Filezilla 3...等。而它 最主要的特色是，它是"跨平台"的 "Native" GUI toolkit，在各种平台上都可写出使用该 平台內建 Native 原

生图形界面的程式。 在 Windows 上就長得跟其他 Windows 程式一 样，在 Linux 下就使用 gtk+ 的图形界面，在 Mac OS X 下就可以使用华丽的 Aqua 外观 风格，这点是非常強悍。 不像 gtk+ 到其它系統都还是只能用 gtk+ 自己的。缺点是，中 文支援在有些地方会出问题，例如剪贴簿的操作，得自己 patch。但仍然相当推荐，即使是 个庞大的 library，效能依旧不会太差，尤其在 Windows 上执行速度并不输 MFC，与其学 MFC，不如学 wxWidgets。

3.最后看看 Qt

Qt 的功能，应该是这三者加上 MFC 之中最強大的，文件也很完整，又有 RAD 工具可 以辅助开发，并且有商业公司做強力后盾。不但有 Windows/X Window/Mac 版本，甚至还有 嵌入式系統可用的版本，稳定性还不错，物件封裝也算良好，资源比 GTK+ 或 wxWidgets 多得非常多，而且发行公司提供了相当多范例，算是一家以开放原始码成功赢利的模范公 司。 知名的 KDE 整个是用它开发，证明了它的稳定性和強大功能。缺点是如果你用它开发 非 GPL 开放程式码的软体，必須以极昂贵的金额，购买商业版本。 而它的图形界面并不完 全是 "Native GUI"，只是透过 theme 去模拟系統上的标准 GUI，所以看起來很像，却会有 些地方可以明显看出破綻。 执行速度缓慢还有过于庞大则是另一个问题。 虽然封裝得很良 好文件也齐全，并不代表他就很容易学还有一个严重问题是，它写的不是标准 C++，它使用 的 signal/slot 机制必須透过 Qt 提供的 preprocessor 处理过才可以转送給编译器，这 部份可能被限定用 qmake，算是一个可惜的地方，不过瑕不掩瑜，还是很推荐。忘了说，它 內部也是 unicode，多国语言没问题。

以上三套只是简单介紹，其中 Qt 的程式我沒有实际完整开发过，但明显的三套都远 远比只能在 Windows 上用，功能少 Bug 多，难学难用，几乎无多国语言支援的 MFC 要 強。三套可跨平台的 library 大家可自行选择，只能用在 Windows 的 MFC 就不用考虑 了。

这篇短评发表出来有些时日了，其中的有些观点（比如 Qt 的授权问题）已经落后于现 在的 Qt 的发展，但其中的大部分论据仍然极有说服力，清晰而简明的指出了几大类库的优 缺点，读者朋友可以从中获益。

1.9 问题与解答

问：LGPL 是什么，和 GPL 有什么区别

答：GNU 宽通用公共授权（GNU Lesser General Public License，或 LGPL）是由自由软件基金会（Free Software Foundation）发布的一个免费软件授权。它被设计作为在 strong-

copyleft GNU 通用公共授权，或 GPL，和简单授权的授权（例如 BSD 授权和 MIT 授权）之

间的一个妥协，LGPL 对程序本身设置了 copyleft 限制，但这些限制不适用于仅与该程式连接的其他软件。

这两个授权都是比较复杂的，需要仔細阅读才可以领会他们重要的区别。总的来说， GPL 要求任何衍生工作成果，比如使用 Qt 开发的应用程式都必须根据 GPL 条款重新進行授 权；而 LGPL 限制性更低，允许开发闭源应用程序。目前最新的 Qt4.5 在 LGPL 2.1 版本下 发布 Qt，并继续在 GPL 3.0 版本下提供 Qt。

问：请问版主，商业版 Qt 提供有服务，而我们开源版遇到自己不能解决的问题怎么办？

答：如果你觉得发现的问题确实是 来自于 Qt 本身，那么你也可以把它提交给 Qt 的。下面是网址：

http://www.qtsoftware.com/forms/feedbackform.html?cid=15 再者，就是你可以把你的问题发到 Qt 的论坛里面，请大家一起讨论解决。关于 Qt 的论坛，在本书的附录中有详细的介绍，可以参考。

问：从哪里可以购买到 Qt 商业版

本人想使用商业版 Qt，主要是与 msvc.net 集成的版本，不知道在哪里可以购买到，在 国外和国内的价格是否有不同呢？

答：个人买的话，在下面的网页上你可以查到价格： http://www.qtsoftware.com/products-cn/pricing-1 你也直接和北京办事处联系一下看如何购买，网址如下：
http://www.qtsoftware.com/about-us-cn/contact

据了解,目前国际上 Qt 的价格是统一的,不存在不同地域价格有差异的问题。 在下面的网页上你可以了解到 Qt 的定价体系以及详细的支持信息，了解这些对你使用

商业版有很大的帮助：

http://www.qtsoftware.com/products-cn/pricing-1

问：Qt4.5 LGPL 版与商业版有何区别？

现在 Qt4.5 支持 LGPL 了,应该就是说不用开放源代码,也可以免费做商业用途了是吧? 但看它主页上还有个商业版本的,那个又是做什么用的?难道说用 LGPL 是不能用作商业的?

答：这两者是不一样的。要明确的是 LGPL 版的 Qt 也是开源版的一种，只是它遵循的协议与 GPL 略有不同，简单来说，LGPL 允许你在不改变 Qt 源代码的条件下，开发商业应用程序而不必开放自己的源代码，这实际上是为 Qt 的商业应用扫清了最后的障碍。

具体讲 LGPL 和商业版的主要区别如下：

首先是 LGPL 版的没有某些商业版拥有的功能(比如 ActiveQt)，同时要求你不能改变Qt 的源代码的情况下才行；商业版的则可以改变 Qt 的源代码。此外，商业版有商业数据库驱动支持，比如 Oracle 等；运行于 Windows 的商业版有 ActiveX 支持。还有就是商业版有售后服务的支持，LGPL 版没有。

问：我想问一下，看到有些书上说只有 Qt4 的商业版才可以和 Visual Studio 结合使用，开源版只能用 MinGW,是这样吗？

答：首先你说的“结合使用”是要看程度的，Qt4 开源版可以使用 Visual Studio 作为 编译器，但不能集成 Qt 工具套件（有些特殊方法可以做到，但笔者不推荐）。要集成使用 的话，需要使用 Qt 提供的 VS Intergration，但它只提供给商业版。

问：我使用开源版连接 sqlserver2005，用 QODBC 的，弹出提示说“Driver not loaded”，请问非商业版是不是不能连接 Sql Server？

答：是的，默认情况下，非商业版没有提供对 SQL Server 的驱动支持，但是非商业版 可以使用某些免费的第三方 ODBC 模块来连接 SQL Server。

问：请问我在商业版中编写的代码能在开源版下编译通过么，反过来呢？

答：从 Qt 的设计上看，应该是可以的，“一次生成，到处编译”是 Qt 的本质特色。 但有一个前提，就是首先要确认你的代码没有使用只有在商业版中才有的功能，然后确定你 的商业版和开源版的对应关系，最好是版本号取得一致 。最后，你在开源版中试一下，如果 可以编译并成功运行，就没有问题。反过来的话，应该是只需要对应好版本就可以了。

问：大家谁知道 Qt 授权中的“专属应用程序”是什么涵义？请指教

如题，在 Qt 官方网站上关于 Qt 授权的说明中写道：“如果您想使用 Qt 开发专属和/ 或商业软件，但又不想共享源代码，那么 Qt 商业授权是正确的授权方式”。

那么这里的“专属应用程序”或者是“专属软件”是什么意思，与商业的有什么区别呢？

答：专属是指的自有知识产权， 那个英文单词在这个语境下确实是不太好翻译，可以

简单的理解为不开源。原英文解释如下：

The Qt Commercial version is the appropriate version to use for the development of proprietary and/or commercial software.

专属软件就是上面那个 proprietary 的，其实应该是拥有产权的，私有的。 我更倾向 于理解为软件版权的归属是私有的，而不是软件的使用范围是私有的。

问：还有一个问题，也是 Qt 的授权中 提到的："运行时收费"，又是什么含义呢？比如商业版，已经通过购买 License 了，为何还要运行时收费呢？

答:运行时收费就是每卖出一份软件拷贝，收取一份费用。简单的说就是按装机量收费。

1.10 总结与提高

Qt 是跨平台应用程序和 UI 框架，可用来编写应用程序，无须重新编写源代码，便可 跨不同的桌面和嵌入式操作系统进行部署。自 Qt 4.5 发布以来，秉承着 Nokia——不断开 发出更多令人喜爱的产品与体验的战略理念， Qt 产品家族不断增添新的举措，新增的授权 选择和新功能使 Qt 比以往更开放、更易用以及更便捷。

本章内容比较丰富，我们向大家介绍了 Qt 的发展历程、Qt 产品的概况以及 Qt 的最新 进展等内容。其中的重点是 Qt4.5 所带来的一系列重大变化，请读者朋友注意在相近的方 面加以对比学习，加深认识。比如 LGPL 协议与 GPL 协议的区别、商业版与开源版的不同 等。

在下一章中，我们将就“Qt 的安装和配置”这部分内容做一个比较深入的探讨，基本 上本章列出的 Qt 支持的平台都会涉及到，希望大家在阅读时用心体会。

第 2 章 Qt 的安装与配置

本章重点

- 掌握获取 Qt 的方法
- 了解 Qt 的协议
- 掌握在 X11 平台上安装配置 Qt 的方法
- 掌握在 Windows 平台上安装配置 Qt 的方法
- 掌握在 Mac OS X 平台上安装配置 Qt 的方法
- 掌握在不同的 Linux 发行版上安装配置 Qt 的方法
- 掌握配置 Qt 环境变量的方法

本章说明了如何获取并在你的系统中安装常见的各种版本的 Qt。这些版本可用于 Windows、X11 和 Mac OS X。可用于 Windows 和 Mac OS X 的预编译库中包含了 SQLite 和 SQLite 的驱动程序，SQLite 目前已用于公众数据库中。从源码包中编译而来的版本则可以 自由选取是否包含 SQLite。开始之前，请先从 http://www.qtsoftware.com/downloads-cn 下载最新版的 Qt。如果你打算开发商业软件，那么就需要购买 Qt 的商业版，然后按照他们 提供的安装说明安装即可。

2.1 获取 Qt

一般说来，可以通过两种方式获取 Qt：一种是开源形式， 另外一种是商业形式。开源 版的各个版本都可以免费获取，而商业版的各个版本则需要通过一定的费用购买得到。

获取开源版 Qt 大致有以下途径： 1.从官方网站上获取最新版本 网址是：http://www.qtsoftware.com/downloads-cn 2.使用网友已经编译好的 Qt 库

目前国内几个主要的 Qt 网站和论坛大多有这方面的内容，如：

- Qt 中文论坛：http://www.qtcn.org/bbs/
- Qt 知识库：http://www.qtkbase.com/
- Qt 核心技术论坛：http://www.insideqt.com/bbs/
- 酷享 Qt 论坛：http://www.cuteqt.com/bbs/

3.从 FTP 下载

如果想使用以前的 Qt 版本，可以从这里下载：

- ftp://ftp.qtsoftware.com/
- ftp://ftp.trolltech.no/qt/source/
- ftp://ftp.ntua.gr/pub/X11/Qt/qt/source/

获取商业版可以从网上购买，网址是 http://www.qtsoftware.com/downloads-cn，也可以直接和 Qt Software 公司联系。

2.2 协议说明

如果你希望发布给予 Qt 开源版创建的那些应用程序，那么就必须遵从在创建这个应用 程序时所使用的 Qt 软件协议中列举出的那些特定条款和条件。对于开源版，这些条款和条 件包括了使用 GNU 通用公共协议（GPL，General Public License）以及 LGPL 协议（Lesser General Public License）的要求。像 GPL 这样的开放协议会给予这个应用程序 的用户一些特定的权利，包括查看和修改源代码以及重新发布这个应用程序（在同等条款下）的权利。如果希望在发布应用程序的同时不公布源代码（要保持源代码的私有性），或者希望在发布应用程序时使用自己的商业协议条件，那么就必须购买创建该应用程序时所使 用到的那些商业版 Qt 软件。这些商业版允许基于自己的条款来销售和发布应用程序。

许可协议中的这些完整法律条款都包含在用于 Windows，Mac OS X 和 X11 的 GPL 版 Qt 中，其中也包含了如何获得商业版的信息。

在本书的第一章中，对 Qt 采用的协议做了介绍和比较，读者朋友可以参阅。如果想更 多的了解这些协议，请登录 http://www.gnu.org/licenses/gpl.html。

2.3 安装 Qt

下面说明了如何在你的系统中安装 GPL 或 LGPL 版的 Qt。这些版本可用于 Windows、 Mac OS X 和 X11(可适用于 Linux 和绝大多数的 UNIX)。

2.3.1 Qt/X11 的安装

1.使用 SDK 安装

在 linux 上使用 SDK 安装 Qt 是比较简便的，按照安装向导的提示，一步一步下来即可 完成，但其中还是有需要注意的地方，下面笔者给出一个图文安装教程，操作系统是 Ubuntu8.04 版，Qt SDK 是笔者写作时最新的 4.5.2 开源版。

第 1 步：下载 SDK

登录到 Qt Software 网站，下载支持 Linux 的 Open Source 版，你下载到的文件名字 类似于 qt-sdk-linux-x86-opensource-2009.03.bin，大小约 276 MB。

第 2 步：设置文件权限

要想安装，用户必须要对该文件有读写和执行的权限。方法是在你下载到的文件上， 用鼠标右键点击，在弹出的对话框上选择 “属性”，如图 2-1 所示，进入到属性设置对话 框。



图 2-1 第 1 步：右键点击，选择属性

接下来，在属性设置对话框中，如果愿意所有用户都拥有读写权限，则可以将所有下 拉框内容选择为“可读写”，并选中【可执行】复选框，然后点击【确定】按钮，完成权 限设置。图 2-2 显示了这个过程。



图 2-2 第 2 步：设置文件的权限

当然，你也可以采用命令行的方式，完成文件权限设置，命令如下：

```
$ chmod 755 qt-sdk-linux-x86-opensource-2009.03.bin
```

第 3 步：开始安装

用 cd 命令切换到你这个文件所在的目录，运行安装程序，方法是：

```
$ ./qt-sdk-linux-x86-opensource-2009.03.bin
```

在有的发行版里面，直接双击这个文件就可以开启安装进程，比如 Red Flag 和 Qomo（原 Everest），赞一下，确实方便用户；而其他多数发行版比如 Ubuntu、Fedora Core、OpenSUSE、Mandriva 等，默认是不可以这样的。你还是老老实实的执行上述命令吧，否则他们会提示你这个文件是不识别的执行文件格式，或者干脆告诉你这个文件不完整， 好多朋友遇到了这个情况，还误以为下载文件不成功的缘故。

好了，安装过程开启后应该如图 2-3 的情形，先解压。

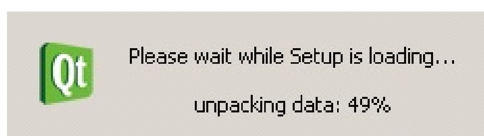


图 2-3 开启安装进程

第 4 步：开始安装

这一步没有什么好说的，安装程序解压缩完毕后，将来到图 2-4 这个欢迎画面，点击【Next】按钮进入下一步。

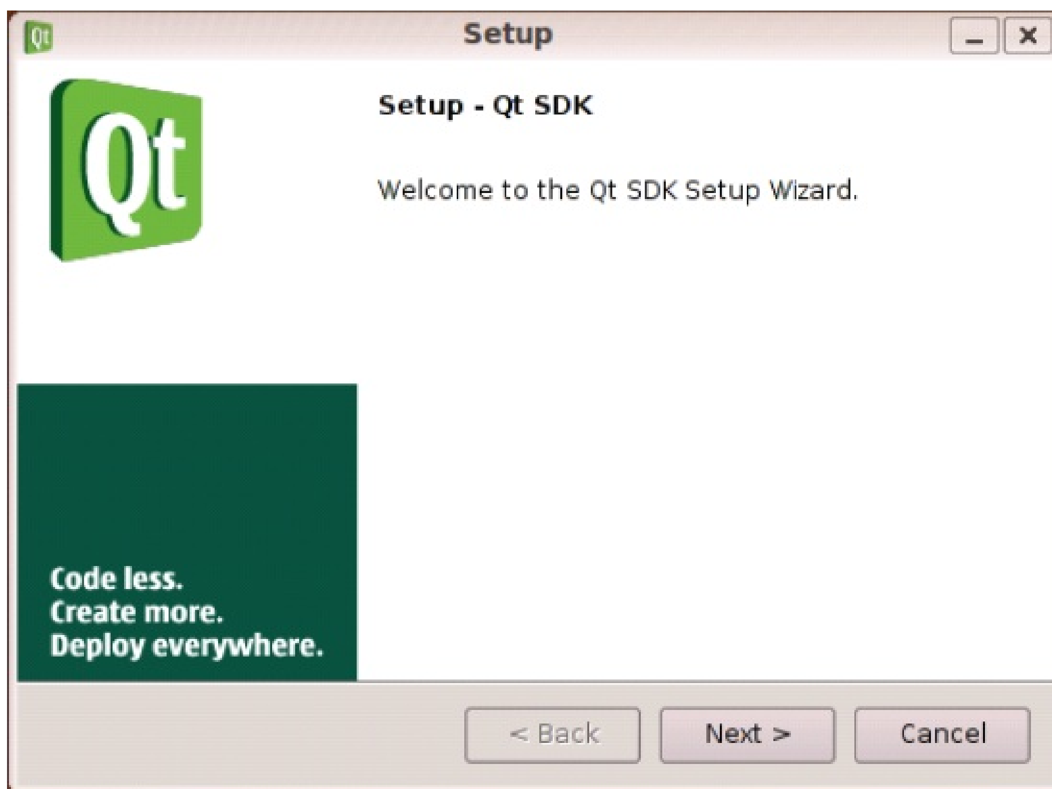


图 2-4 开始安装

第 5 步：接受授权协议

在图 2-5 这个画面，安装程序将对 Qt 采用的协议做一个说明，阅读完毕后，选择接受 协议那个选项，然后点击【Next】按钮进入下一步。



图 2-5 接受协议

第 6 步：选择安装路径

这一步是配置 Qt 的安装路径，如果没有特殊要求，建议选择默认路径。当然你也可以 修改，但是切记一点，就是不可以在路径中加入空格，否则以后很可能出现意想不到的问题。其实，如果你对研究了解一下 Qt 自身对变量、路径等的命名方式，就会发现它们都是不带空格的，如果确实需要隔开的话，就用 -好了。选择好之后，在图 2-6 中点击【Next】 按钮进入下一步。

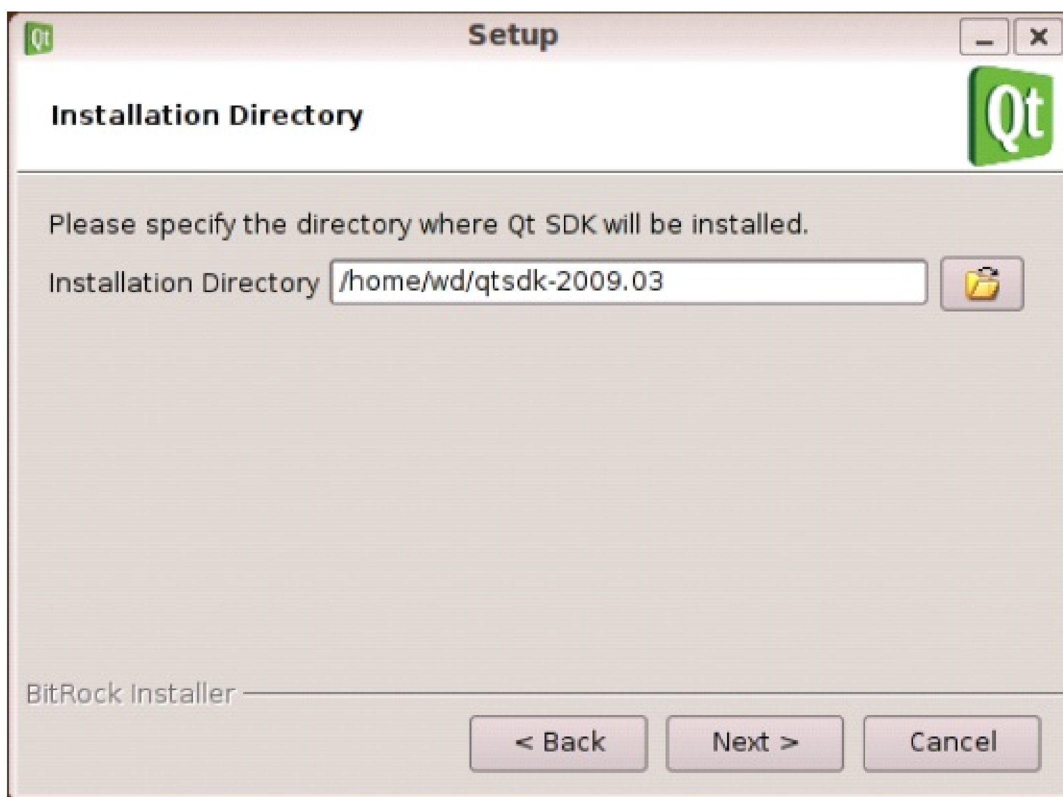


图 2-6 选择安装路径

第 7 步：选择安装组件

在图 2-7 所示的选择安装组件对话框中，有两个选项，其中 Qt Creator 是必选的，Qt Development Libraries 也是建议选择，否则我们是干什么来了，就是要使用 Qt 的库阿，点击【Next】按钮进入下一步。

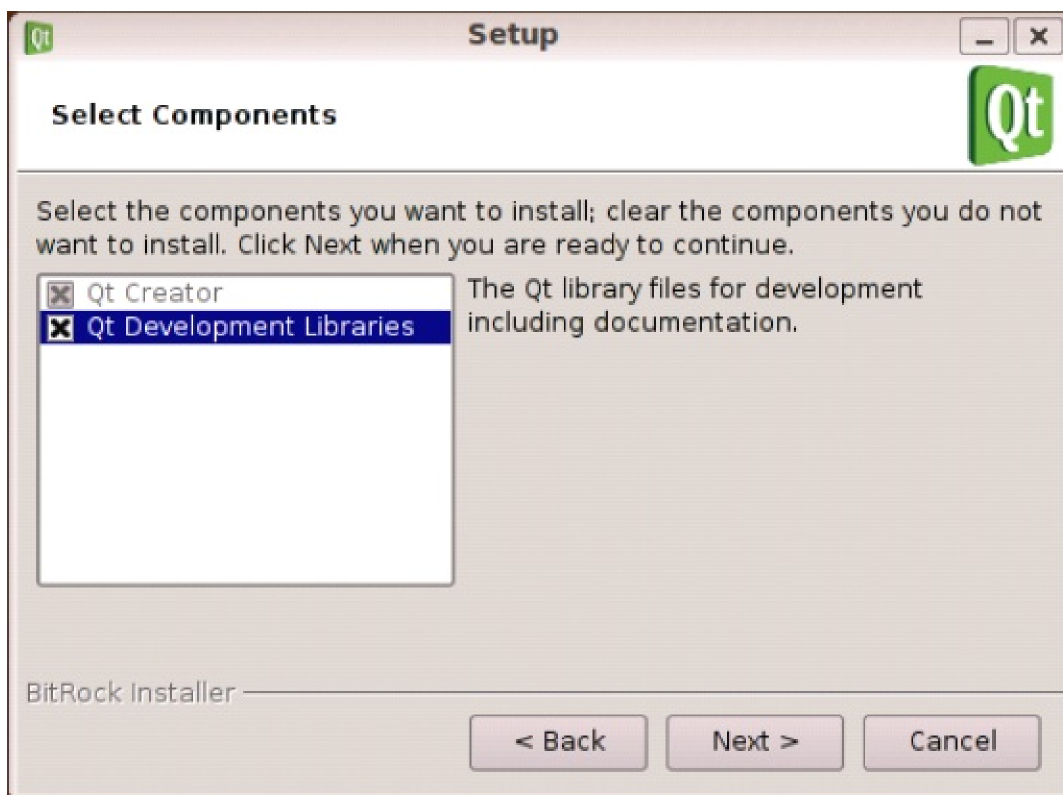


图 2-7 选择安装组件

第 8 步：开始安装

这一步没有什么好说的，在图 2-8 上点击【Next】按钮开始安装。

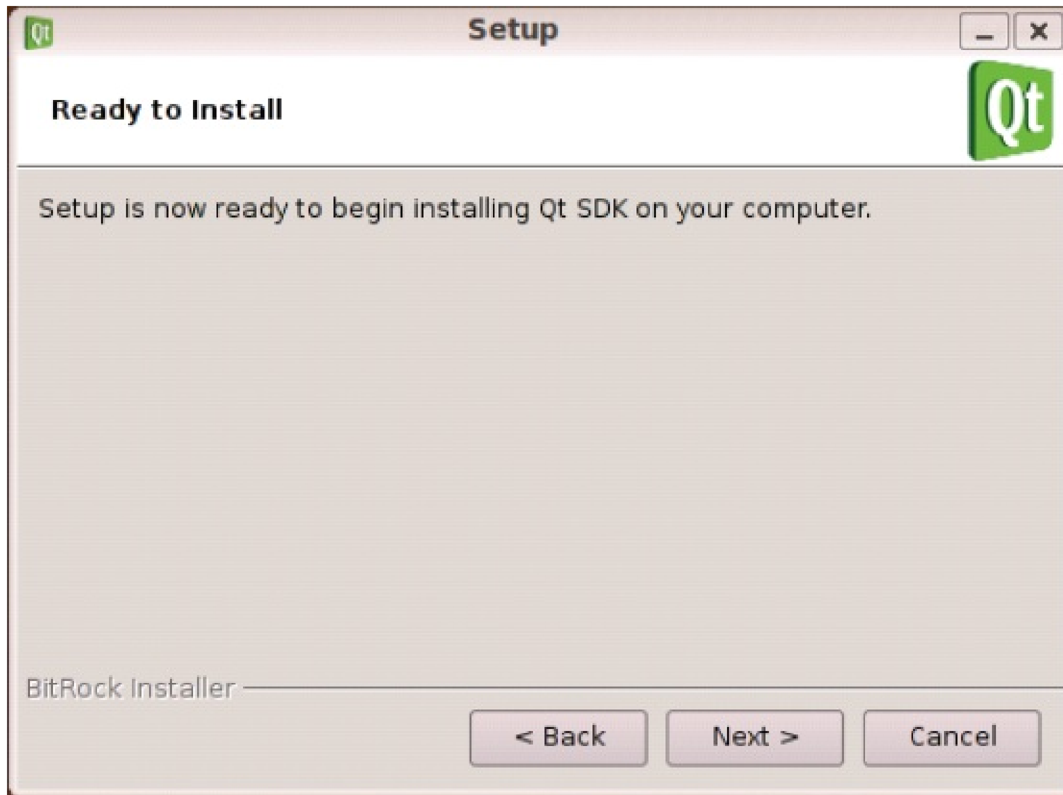


图 2-8 准备开始安装

第 9 步：安装过程

这一步我们做看客，静静等候安装结束，这个过程大约需要若干分钟。如图 2-9 所示 的画面显示了详细的安装进度和内容，如果想取消，可以点击 【Cancel】按钮。

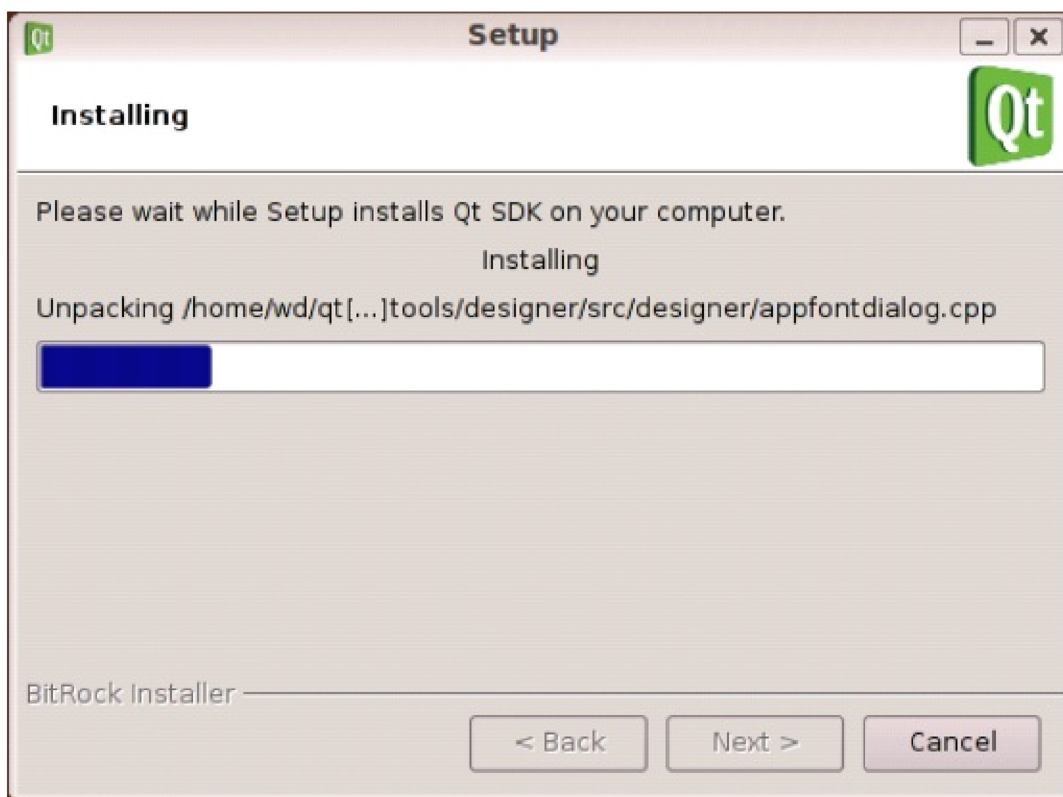


图 2-9 安装进行中

第 10 步：安装过程结束

如果到了图 2-10 这一步，并且中间没有提示错误，那么安装过程就结束了，选中那个 复选按钮，以验证是否成功。

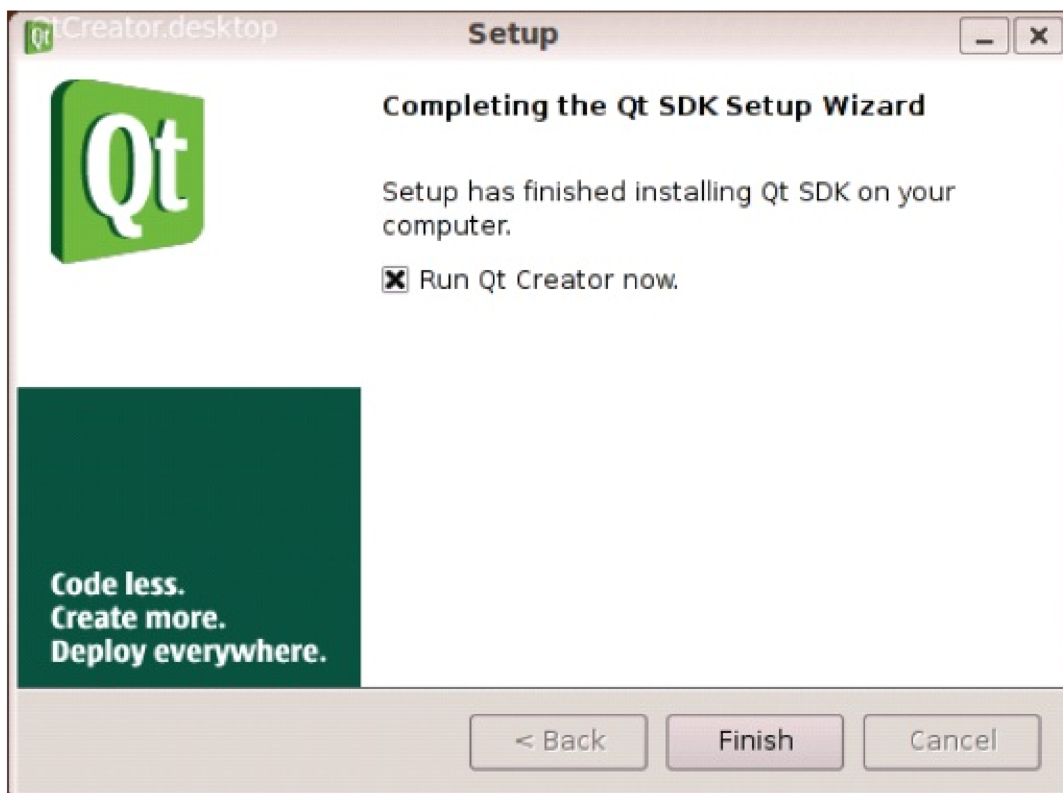


图 2-10 安装成功

第 11 步：验证安装

一切正常的话，如图 2-11 所示，Qt Creator 将会来到你的面前，表示安装成功了。 Qt Creator 是 qtsoftware 官方出品的 IDE，以后我们还会讲到它。

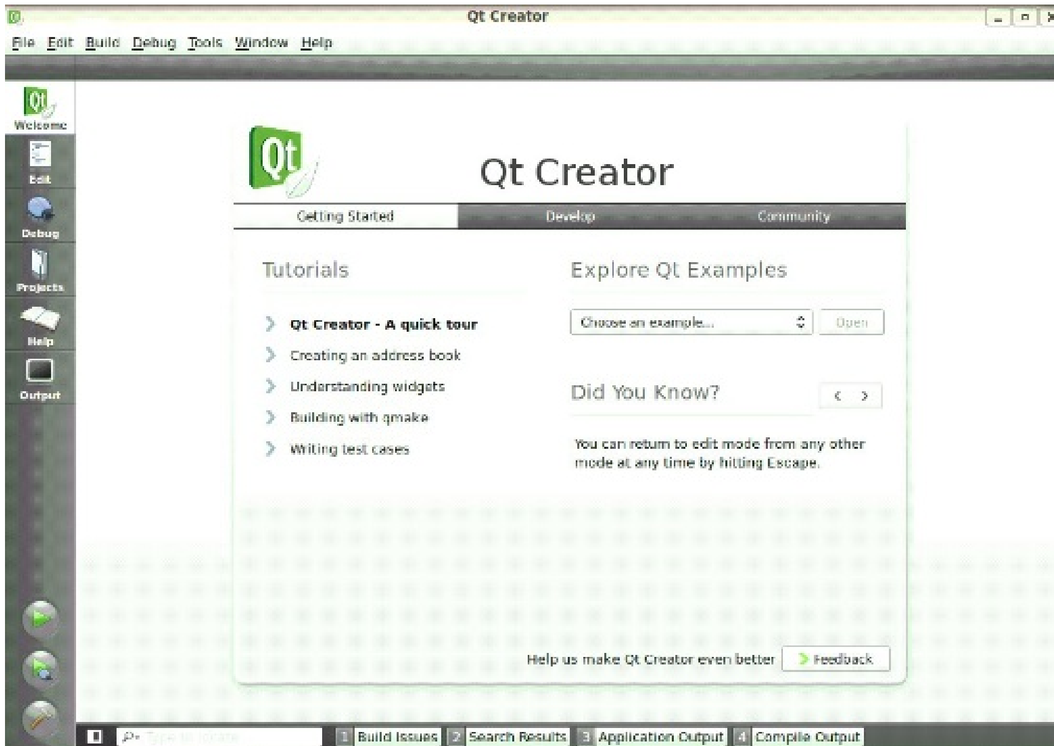


图 2-11 运行 Qt Creator 以验证安装

第 12 步：配置 Qt 环境

如果你今后完全使用 Qt Creator 来编程的话，不配置 Qt 环境也无妨，因为 Qt Creator 会准确的找到你的 Qt 安装位置。但还是建议配置一下，因为在很多情况下我们还

是要用到基本的命令。

关于 Qt 环境变量的配置，请参见后面的第 2.4 节。

2.编译安装的方法

从 qt 的网站中下载文件 qt-x11-opensource-src-4.5.2.tar.gz(在写作本书时采用的 就是这个版本，但当你阅读此书时，使用的文件可能已经发生了改变)。在 X11 中，要把 Qt 安装到它的默认位置，需要拥有 root 权限。如果没有 root 权限，那么请使用 config 工具 的-prefix 选项来指定一个你具有操作权限的目录。

第 1 步：把当前路径切换到你存放下载文件的目录处 例如：

```
cd /tmp;
```

第 2 步：解压缩该压缩文件

命令是：

```
gunzip qt-x11-opensource-src-4.5.2.tar.gz

tar xvf qt-x11-opensource-src-4.5.2.tar
```

此时会生成一个/tmp/qt-x11-opensource-src-4.5.2 目录。Qt 需要的是 GNU 的 tar 工具，而在某些系统中它称为 gtar。

第 3 步：运行 configure

在整个的编译过程中，configure 这一步很关键。configure 的作用有两个，一是生成 平台相关的 qmake；二是配置 Qt 的 Feature，比如源文件放在哪里，库在哪里，是否支持 OpenGL 等等。

用你喜欢的选项来执行 configure 工具，它可用于编译 Qt 库以及与 Qt 一起提供的工 具软件：

```
cd /tmp/qt-x11-opensource-src-4.5.2

./configure
```

要查看 configure 的配置选项列表，可以运行./configure -help 命令。

如果 configure 失败，可以以使用-v 选项查看具体的原因，命令是./configure -v 如果以后需要在这次配置的基础上更改选项，先要运行 make confclean 命令以清除之前的配置。

第 4 步：执行 make

这一步是根据 configure 时由 qmake 生成的 makefile 来编译 QT 库。只要前面配置好 了，一般不会出什么问题。

要编译 Qt，输入命令：

```
make
```

这样将会生成 Qt 库，同时也会编译所有的演示程序、示例程序和工具软件。在某些系统中，make 命令称为 gmake。 第 5 步：安装 Qt 输入命令：

```
su -c “make install”
```

然后输入 root 密码。（在某些系统中，上述命令是： sudo make install）这样就可以把 Qt 安装到/usr/local/Trolltech/Qt-4.5.2 目录中。如果要改变安装路径，那么可以 在 configure 命令的后面使用-prefix 选项来做到这一点。如果你已经对安装目录具有写操 作权限的话，那么只

需输入以下命令即可：

```
make install
```

第 6 步：设置环境变量

如果使用的 shell 是 bash、ksh、zsh 或者 sh，那么请把以下两行代码添加到.profile中：

```
PATH=/usr/local/Trolltech/Qt-4.5.2/bin:$PATH

export PATH
```

如果使用的 shell 是 csh 或者 tcsh，那么请把下面一行代码添加到.login 文件中：

```
setenv PATH /usr/local/Trolltech/Qt-4.5.2/bin:$PATH
```

如果使用了 configure 的-prefix 选项，那么请使用你自己指定的路径来代替这里给出的默认路径。

如果你正在使用的编译器不支持 rpath 命令，那么还必须扩展 LD_LIBRARY_PATH 环境 变量，使其包含/usr/local/Trolltech/Qt-4.5.3/lib。对于带有 GCC 的 Linux 用户来讲， 则没有必要执行这一步。

第 7 步：验证安装 在命令行输入：

```
qmake -v
```

看看输出是什么。

举个例子，如果你看到的是如下的输出，则表示你的 qmake 链接指向 Qt3.3.8 的版本 而不是 Qt4.5，你需要检查你的安装是否成功，以及环境变量是否配置正确。

```
[wd@localhost ~]$ qmake -v

Qmake version: 1.07a (Qt 3.3.8)
Qmake is free software from Trolltech ASA.
```

如果你看到如下的信息，则表示你的 Qt 4.5 配置成功了。

```
[wd@localhost ~]$ qmake -v

QMake version: 2.01a (Qt 4.5.2)
QMake is free software from Trolltech ASA.
```

第 8 步：删除源码和配置文件

这一步是可选的，运行 make clean 命令后，可以删除掉本次安装时的源码和配置文 件，这样可以节省空间。笔者做过测试，以安装 Qt4.5 版为例，如果采用 configure - static -release 的参数编译，执行 make clean 后，大约可以节省 500 Mb 的空间。

如果你想以后在不改变原来配置的基础上，多编译一些内容，比如数据库驱动的话， 就不要执行这个命令了。

小贴士：如果你还需要要编译 Qt3 的程序，你最好使用其它的用户进行安装，并建立 Qt3 对应的环境变量。实际应用中，还存在这种情况，就是系统中自带了 Qt3 的包，而你又编 译安装了 Qt4 库，那么使用中最方便的调用 Qt4 库的方法是将命令 qmake 改为输入 qmake- qt4

专题：如何编译数据库驱动

Qt 在采用缺省安装参数进行编译安装的情况下，是不配置安装数据库驱动的，所以你 如果要驱动数据库，就需要在 configure 的时候，配置相应的参数。这方面的详细说明， 可在安装完成后的 Qt Assistant 的中搜索与 SQL Database Drivers 相关的条目查到。

下面说说如何安装 psql, odbc, sqlite, mysql 的驱动。 在 configure 的时候，要加上配置项，命令如下：

```
./configure -plugin-sql-mysql -plugin-sql-sqlite -plugin-sql-odbc -plugin-sql-psql
```

但是在运行的时候，它会可能会提示你测试不到 mysql 函数。于是，要修改一下文件

```
src/plugins/sqldrivers/mysql/mysql.pro（以上 mysql 的路径请根据你自己的实际情况 调整）。
```

在最开始处，加上下面两行（具体的目录，需要你自己查看系统的实际位置）：

```
INCLUDEPATH += /usr/include/mysql
LIBS += -L/usr/lib/mysql
```

保存后，退出来，然后运行:

```
./configure -plugin-sql-mysql -plugin-sql-sqlite -plugin-sql-odbc -plugin-sql-psql -conti
```

再然后，就和其它的一样了，依次执行：

```
make
make install
```

但是在安装后，你会发现，mysql 的驱动并未在 sqldriver 目录下面，于是再转到

src/plugins/sqldrivers/mysql/目录下，运行：

```
make
```

然后把 plugins/sqldrivers/目录下面的 libqsqlmysql.so, libqsqlmysql_debug.so

这几个文件拷贝到/usr/local/Trolltech/Qt 4.5.2/plugins/sqldrivers 目录下面，这样 就可以了。

可能有的同学会问到，如果自己已经按照默认的参数配置安装完了 Qt4，但又想使用数 据库驱动，是不是要重新编译一次呢？从理论上来说，是要重新编译，但实际上不用重新编 译，这是由 make 机制决定的，只要你 make install 后源代码及编译后的东西没有删除， 也就是没有使用 make clean 命令的话，编译的时候 qmake 会自动的略过已经编译过的东 西，而只编译另外附加的数据库驱动。

2.3.2 Qt/Windows 的安装

从 4.5 版以后，Qt 的安装有了多种选择，你也可以像以前那样，从源代码包编译安 装；也可以用 Qt 最新提供的 SDK 安装程序安装。

1.使用 Qt SDK 安装

下面笔者给出一个 Qt4.5 opensource 版 SDK 安装的图文过程，操作系统是 Windows XP SP2 中文版，Qt 版本是 qt-sdk-win-opensource-2009.03 版，这一过程在 Windows Vista 上也是类似的，请大家参考使用。

第 1 步：下载

这一步不用多说了吧，到 qtsoftware 网站，如果英文不熟悉，先切换到中文版，方法 是在右上角点那个五星红旗，如图 2-12 所示。然后在下载链接里面选择 windows 平台上对 应的版本，就是下载 Qt SDK for Windows* (194 Mb) 这一项，你下载到的文件名字类似于 qt-sdk-win-opensource-2009.03.exe。

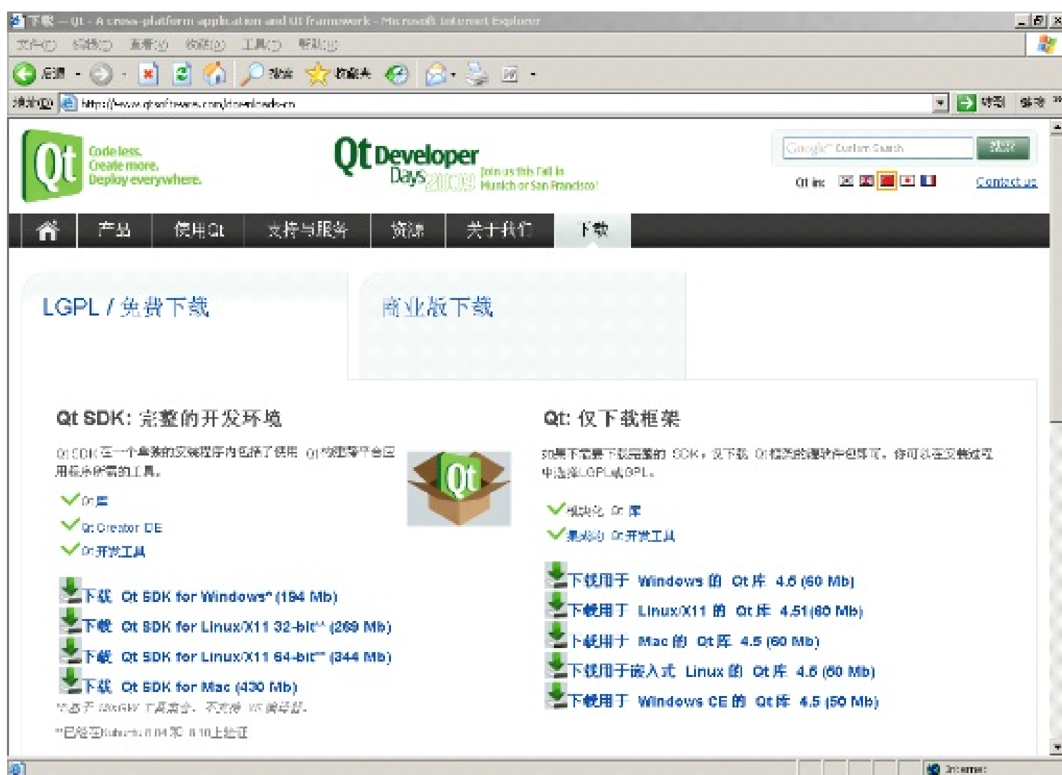


图 2-12 下载 Qt SDK for Windows

第 2 步：在本地双击安装 安装程序将先解压，这个过程大概有若干分钟，视你的机器配置而有不同，程序自解压完成后，将进入下一步，如图 2-13 所示

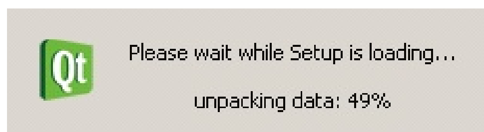


图 2-13 在本地双击安装

第 3 步：同意遵守授权协议

在下图 2-14 中同意遵守授权协议，或者按下 Alt+A 键即可，点击【Next】按钮，进入 下一步。

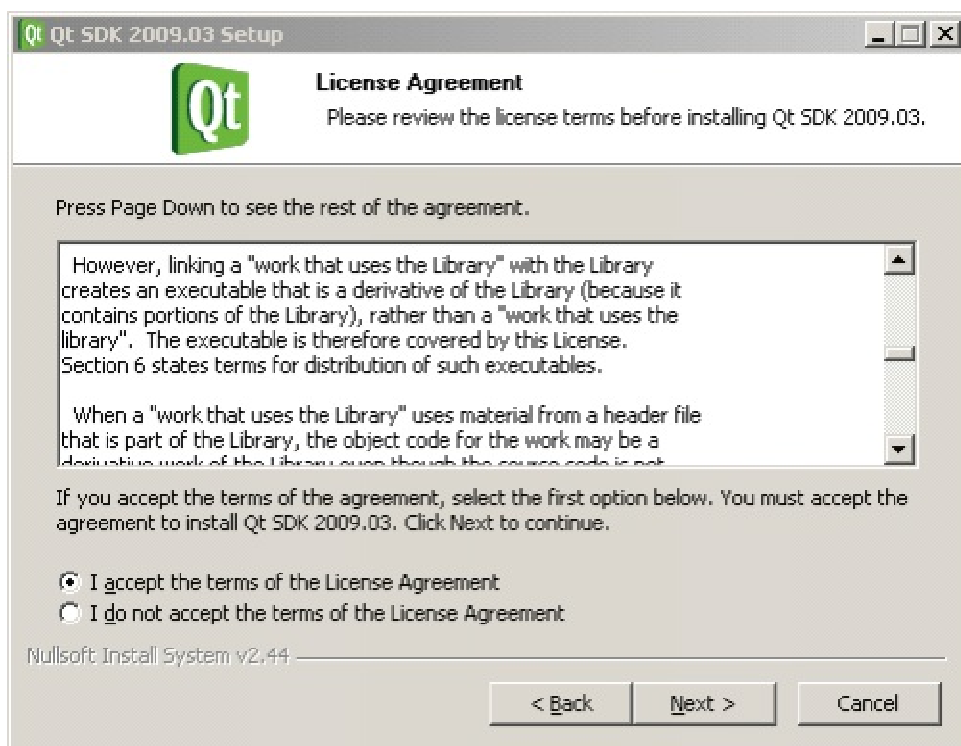


图 2-14 遵守 Qt 协议

第 4 步：了解与 Qt 集成的调试器（debugger）的内容

如图 2-15 所示，其内容大意是说，默认 Qt 将选择 gdb 作为调试器，它是集成到 MinGW 里面的，MinGW 会随着本安装程序一同安装；而如果要使用 CDB 作为调试器的话，需要到微软的网站上取得，因为使用 VS 创建的项目将使用 CDB 作为调试器。此外，该版本的 Qt Creator 目前只支持 32 位调试模式。



图 2-15 了解 Qt 调试器的内容

第 5 步：选择安装组件

如图 2-16 所示，默认情况下，安装程序推荐你安装所有 5 项组件中的前 4 项，其中第1 项是默认必需的，我们建议遵从默认的选择；点击【 Next】按钮进入下一步。

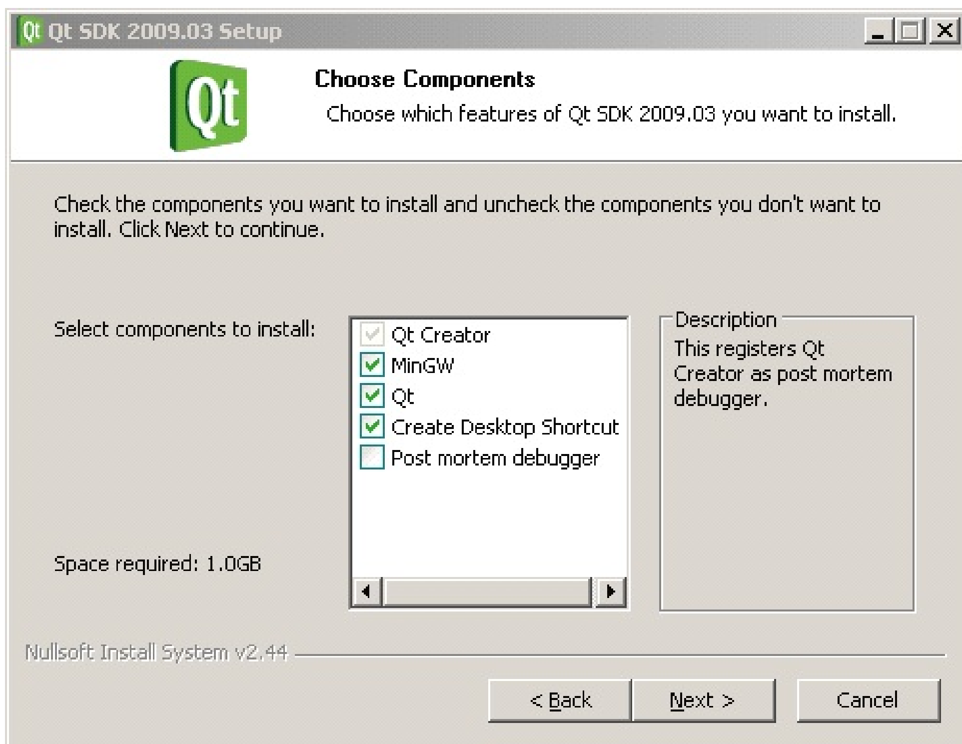


图 2-16 选择安装组件

还有要说明的是最后一个选项：Post mortem debugger，它的意思是程序崩溃后调试的工具，类似 coredump 之类的。注意，post mortem 是一个结合在一起才有意义的词。举 个例子，有时候程序崩溃了，会问你要不要用 visual studio 调试。选上这个框之后，在应用程序崩溃时，就可以调用 Qt Creator 直接调试了，所以你也可以把它选择上。

第 6 步：选择安装路径

这一步比较容易，大家可以根据需要调整 Qt SDK 的安装路径，然后可以点击【Next】 按钮进入下一步，如图 2-17 所示。在这里仍然提醒路径中不要包含空格和特殊字符。

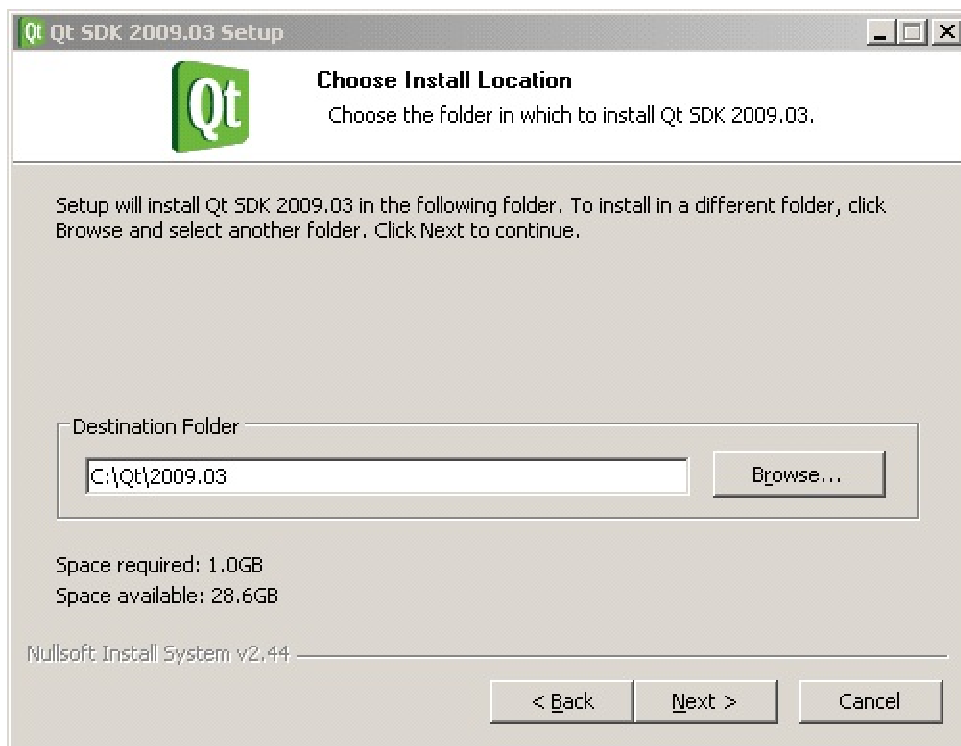


图 2-17 选择安装路径

第 7 步：调整快捷方式的名称

这一步也容易，你可以根据需要调整安装后在程序菜单上显示的快捷方式的名称，然后点击【Next】按钮进入下一步，如图 2-18 所示。

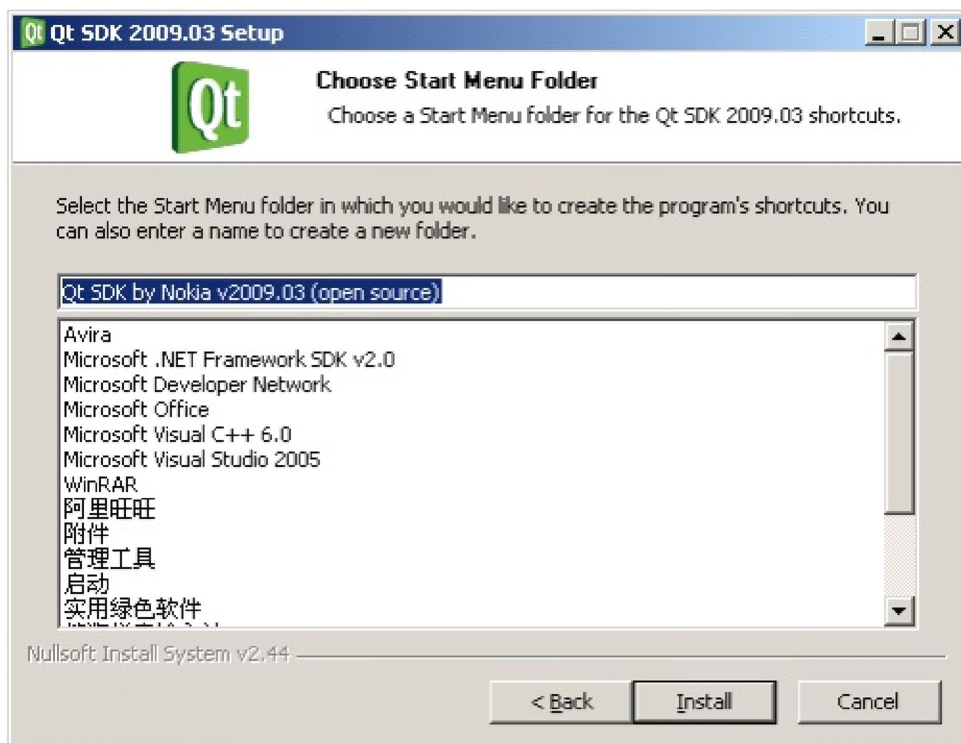


图 2-18 调整快捷方式的名称

第 8 步：安装过程进行中

到这一步后，你就只能做看客了，静静等待安装过程进行。如果想看仔细，就点击那 个【Show details】按钮，如图 2-19 所示。

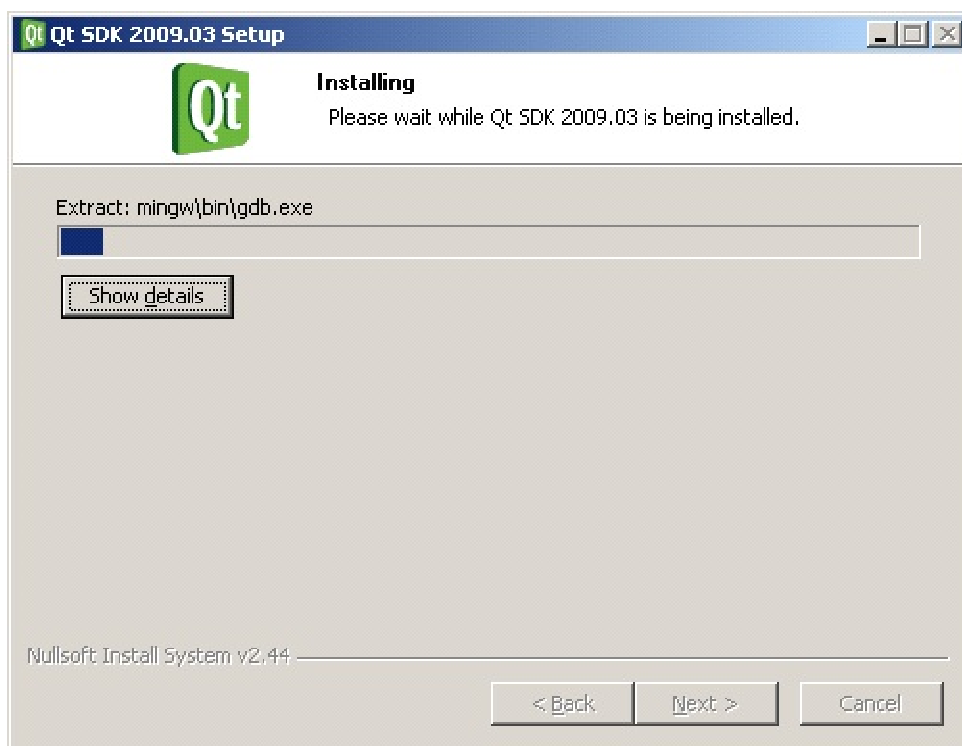


图 2-19 安装过程正在进行

第 9 步：安装过程告一段落

通常安装到这一步，并且中间没有弹出什么错误提示之类的情况，就基本成功了，点 击【Next】按钮进入下一步，如图 2-20 所示。

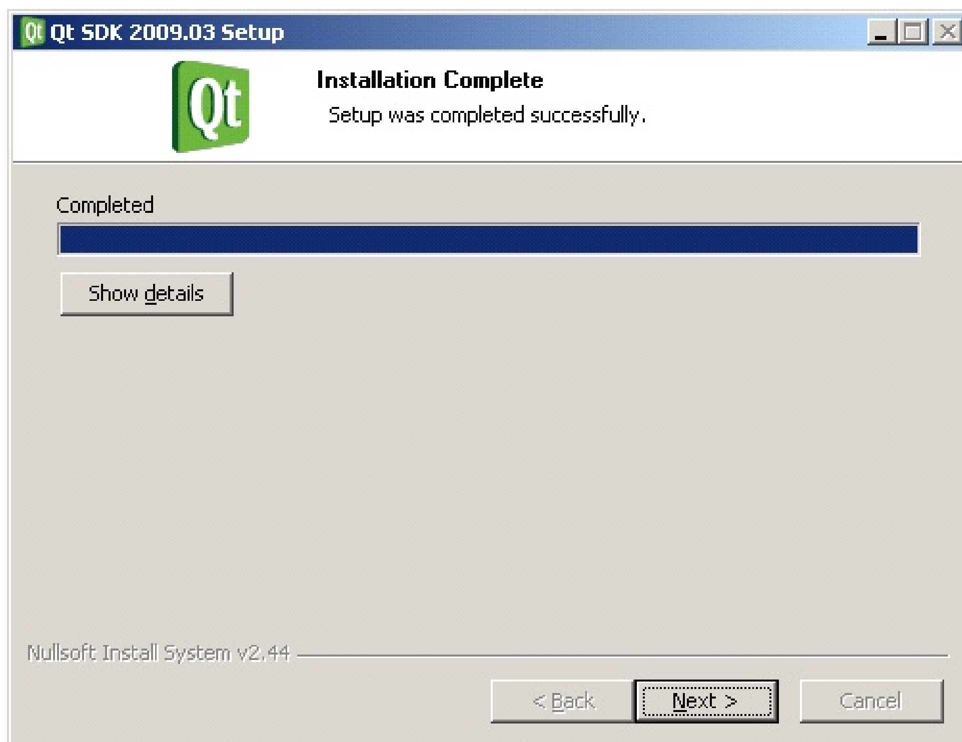


图 2-20 安装过程告一段落

第 10 步：安装结束

安装程序会提示你安装过程成功结束，并且建议你立即运行 Qt Creator 验证和体验一 下，点击【Finish】按钮，如图 2-21 所示。



图 2-21 安装结束

第 11 步：验证安装，运行 Qt Creator

一切顺利的话，Qt Creator 会来到你的面前，请看图 2-22。



图 2-22 运行 Qt Creator 成功

好了，安装过程到此结束，接下来还是需要配置环境变量，请参阅后面的章节。

2.从源代码编译安装

第 1 步：准备工作

首先就是确定你要安装的 Qt 版本和需求，如是 OpenSource 还是 Commercial 版，准备与 MinGW 集成还是与 VS 系列集成（这方面内容在后面第 4 章会讲到）等等。下面的几个原则很有用：

- 在 windows 上编译安装 Qt 最好遵循一定的顺序 首先安装操作系统，再安装你选定的编译器 或 IDE，然后再安装 Qt。
- 安装操作系统时不要选择精简版或 home 版等 因为某些库环境不完全，可能会出现问题。
- 创建一个干净的系统环境

在我们这里是与 MinGW 集成，所以就先安装它，不要安装其他的编译器。 举个例子，如果你的计算机上安装了 Borland C++ Builder， 在它的目录下有个make.exe 文件，系统可能会把 MinGW 下的 make.bat 文件搞错，从而导致不能正常编译。

做好了上面这些，就可以开始安装了。 第 2 步：安装 MinGW

MinGW 是 Minimallist GNU for Windows 的缩写，是在 Windows 上的 GNU 工具集。在 Windows 上安装开源版 Qt，首先需要安装 MinGW。安装 MinGW 有几种方法，一种是从网站上 下载安装，一种是通过安装 dev-cpp 来安装 MinGW。

第一种方式：

首先是下载 MinGW，有几个可以下载 MinGW 的网站： 官方网站： http://www.mingw.org/

sourceforge 的网站：http://mingw.sourceforge.net 本书写作时的使用的版本是最新的 MinGW-5.1.4，你下载到的文件名字类似于 MinGW-5.1.4.exe 或者是 MinGW-5.1.4-2008-12-4.exe，前者实际上是网络安装的一个链接，后者是完整的本地安装版本，大约有 13Mb 的样子。你使用哪一种都可以，速度都够快，安装过 程是一样的。下面就给出关键步骤的图文说明：

然后双击安装程序，出现如图 2-23 所示的欢迎画面。



图 2-23 MinGW 安装程序欢迎画面

单击【Next】按钮，进入下一步的下载安装界面， 如图 2-24 所示。选择第一项，即下 载的同时安装，单击【Next】按钮进入下一步。



图 2-24 选择下载方式

接下来这一步是阅读并同意协议授权，单击【I Agree】按钮，进入下一步。



图 2-25 阅读并遵守协议

这一步是询问你需要 MinGW 的版本，有 3 个选项：Previous、Current 和 Candidate， 分别对应以前的、当前的和预发行的版本，选择 Current 即当前版，单击【Next】按钮进 入下一步。

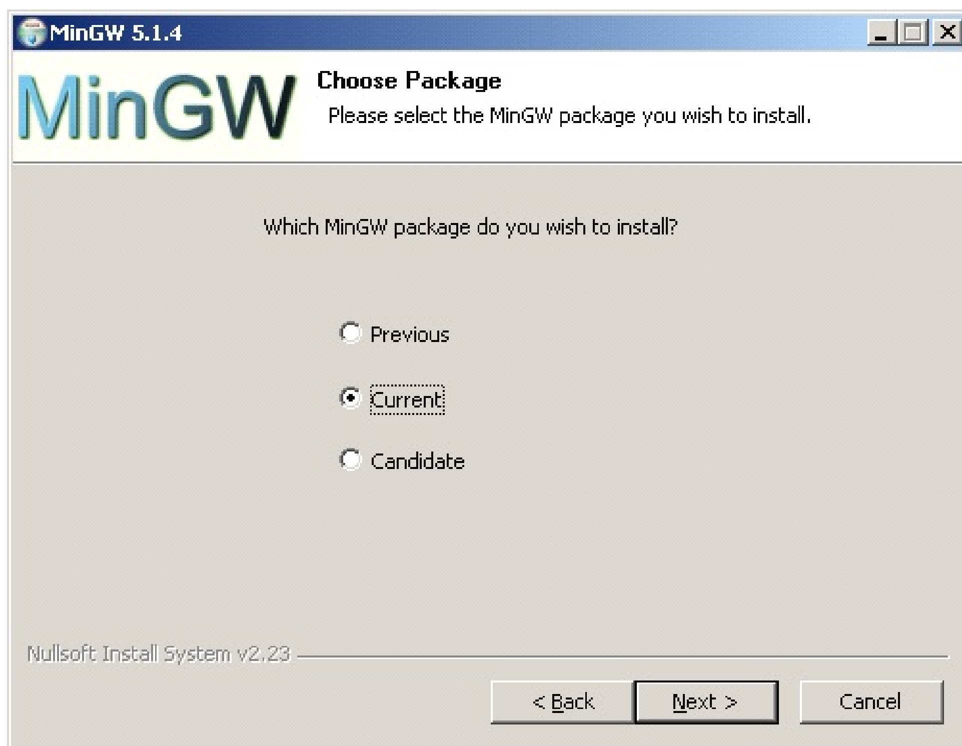


图 2-26 选择安装包类型

如图 2-27 所示，这一步是选择要安装的 MinGW 的组件，我们需要的是 gcc、g++ compiler、MinGW Make 这几个选项。选中后，点击【Next】按钮进入下一步。

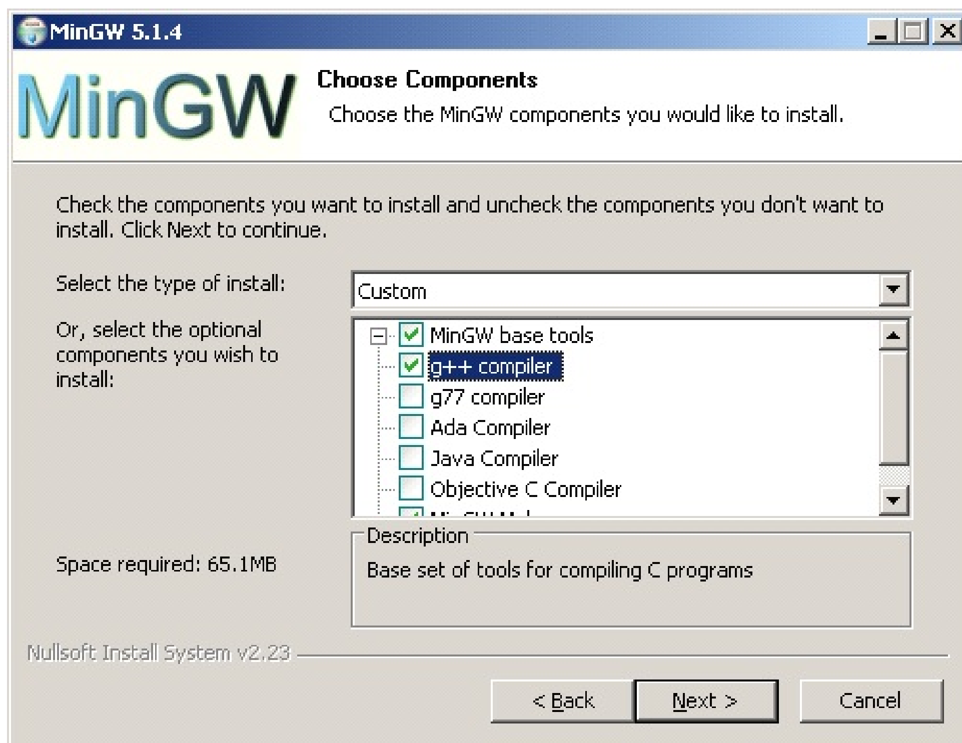


图 2-27 选择 MinGW 组件

接下来的两步分别是选择安装路径和配置快捷方式的名称，选择缺省，一路点击【Next】按钮即可。然后安装程序会开启网络安装进程，如图 2-28 所示。

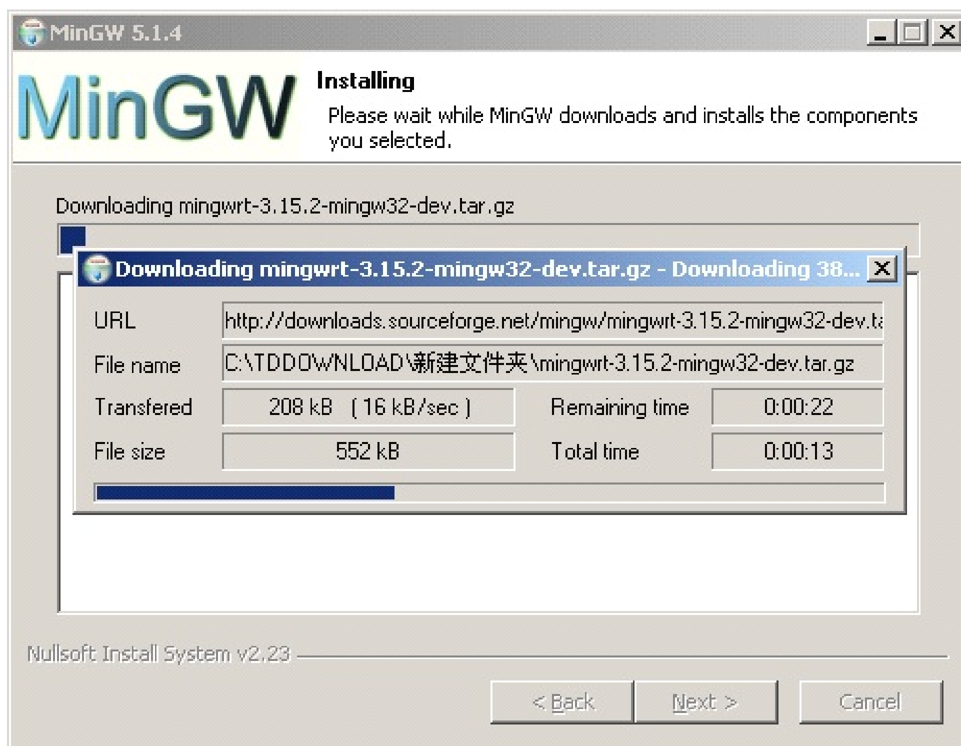


图 2-28 网络安装进程

约莫十来分钟的时间，安装过程顺利结束。

第二种方法步骤如下：

首先是下载安装文件，需要下载 Dev-cpp。Dev-C++是一个 Windows 下的 C 和 C++程序 的集成开发环境。它使用 MinGW32/GCC 编译器 。我安装的是 devcpp-4.9.9.2_setup.exe，Dev-cpp 的官方网站是 http://www.bloodshed.net/， 可以登录上 去下载最新的版本。

然后是执行安装文件。先执行 devcpp-4.9.9.2_setup.exe 文件。其中，需要注意的 是【组件选择】对话框，如图 2-29 所示。请勾选上【Mingw compiler system】。

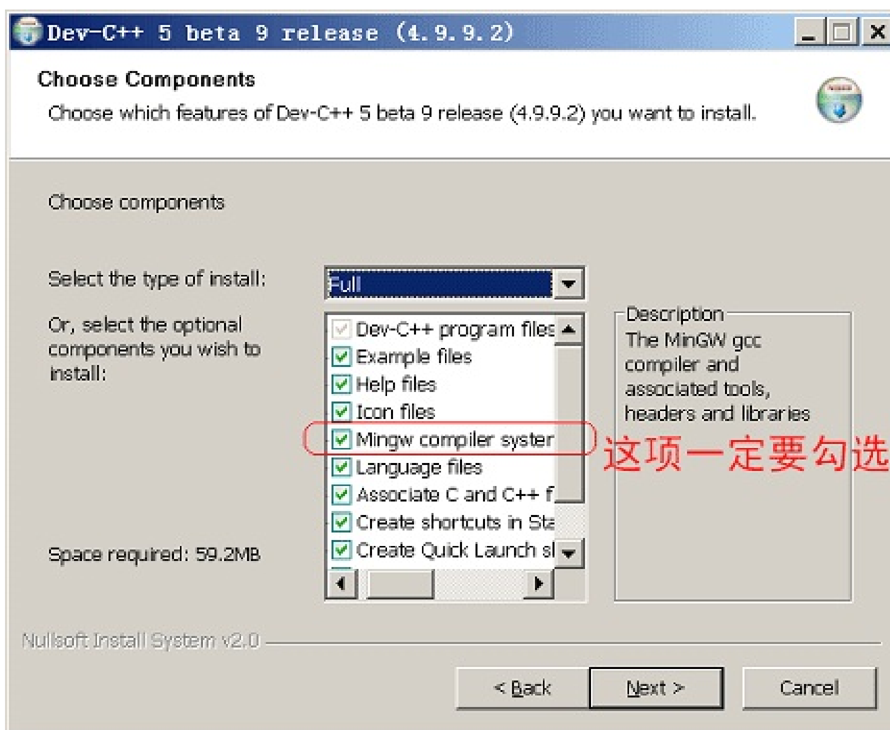


图 2-29 选择安装组件 接下来就是都选择默认选项，一路点击 【 Next】按钮直至安装结束。
第 3 步，下载并解压缩 Qt 源代码

你下载到的源代码文件类似于 qt-win-opensource-src-4.5.2.zip，把它解压缩到你想安装 Qt 的目录，比如是 d:\Qt\OpenSource。请注意， 在 Qt 的应用中，路径大都可以自定,但是 尽量不要在路径中包含空格、中文或其它的特殊字符，否则可能会出现意想不到的问题。

第 4 步，编译 Qt 源代码

运行 cmd 命令进入 DOS 窗口，使用 cd 命令进入你的 Qt 源代码的目录，比如这里是 D:

```
\Qt\OpenSource\qt-win-opensource-src-4.5.2
```

屏幕显示如下：

```
C:\&gt; D:
D:\&gt; cd D:\Qt\OpenSource\qt-win-opensource-src-4.5.2
```

运行 configure 命令进行配置，输入 configure -help 可以获得选项的列表并使用自

定义的选项。比如要同时配置 debug 和 release 版本，可以输入命令 configure -debug- and-release，屏幕显示如下：

```
D:\Qt\OpenSource\qt-win-opensource-src-4.5.2&gt;configure -debug-and-release
```

这一步类似于在 X11 上编译 Qt 时 configure 的情形，大约有十几分钟就可以完成了。

然后运行 MinGW 的 make 工具，命令是 mingw32 –make。 屏幕显示如下：

```
D:\Qt\OpenSource\qt-win-opensource-src-4.5.2&gt; mingw32-make
```

这个过程比较长，大约几个小时。编译成功后，可以手动创建桌面快捷方式如

qtdemo、designer 等。Qt Windows 开源版本不同于 X11 上的版本，不需要安装，编译成功后就可以使用。

第 5 步，设置环境变量

按下 Ctrl+Break 进入系统设置，也可以从控制面板进入，选择 “高级→环境变量”， 显示如图 2-30 所示。

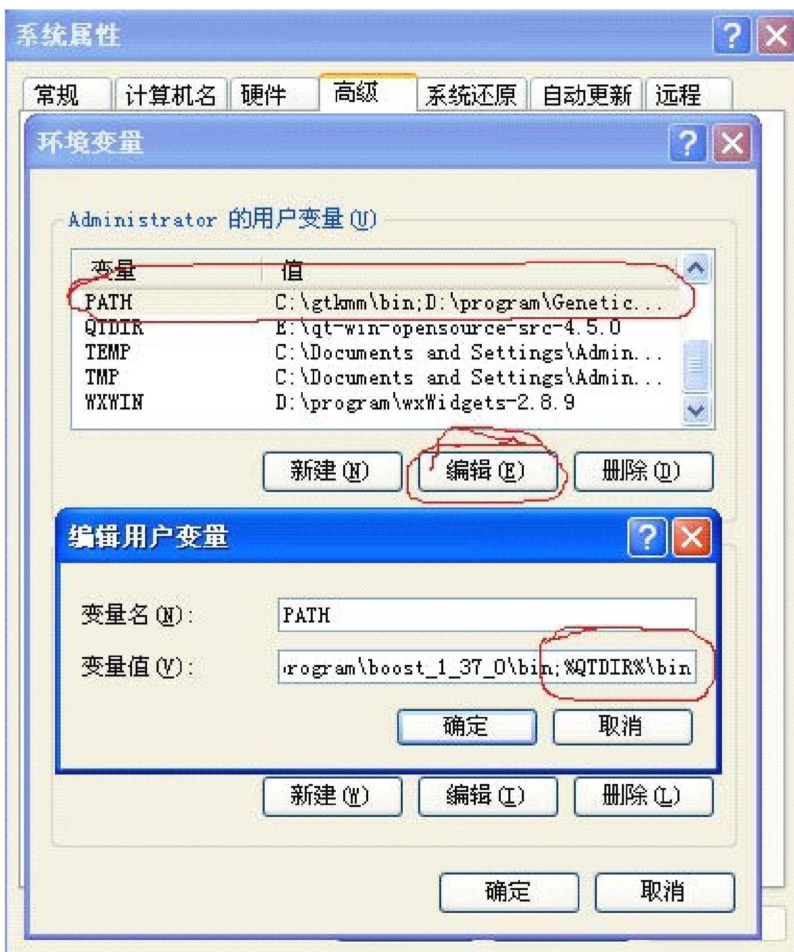


图 2-30 设置环境变量

在用户变量中设置 PATH 环境变量为：d:\Qt\OpenSource\qt-win-opensource-src-4.5.2;%P%ATH。

小贴士：在 Windows Vista 中，需要对 MinGW 进行额外的设置才能正常使用，具体步骤如下：

设置 GCCPREFIX 环境变量为 MinGW 的安装目录，如：C:\MinGW；

设置 PATH 环境变量为%GCCPREFIX%\libexec\gcc\mingw32\4.0.0:%PATH%，其中 4.0.0 的版本号需要根据你的 GCC 版本实际情况调整；

编译 Qt 的步骤与 Windows XP 上相同。

3.从框架安装程序安装

(1) 下载安装文件

在本书还在写作时，Qt 在 Windows 下的安装程序称为 qt-win-opensource-4.5.2-mingw.exe。这个版本号可能与你阅读本书时所看到的版本号有所不同，但其安装过程应当是相同的。要开始安装过程，请先到 qtsoftware 网站上把这个文件下载下来并运行它。

(2) 安装 MinGW

在执行框架安装程序之前，必须先安装 MinGW，安装步骤清参考上一节，这里不再赘述。

(3) 执行安装

双击执行 qt-win-opensource-4.5.2-mingw.exe 文件。开头的几步与使用 SDK 安装过 程类似，直到如图所示的【选择安装组件】这一步，列表中有两项，其中 Qt4.5.2 这一项 默认必选，而“File Associations”这一项是建立.ui 文件与 Qt Designer 的关联，建议 选上。

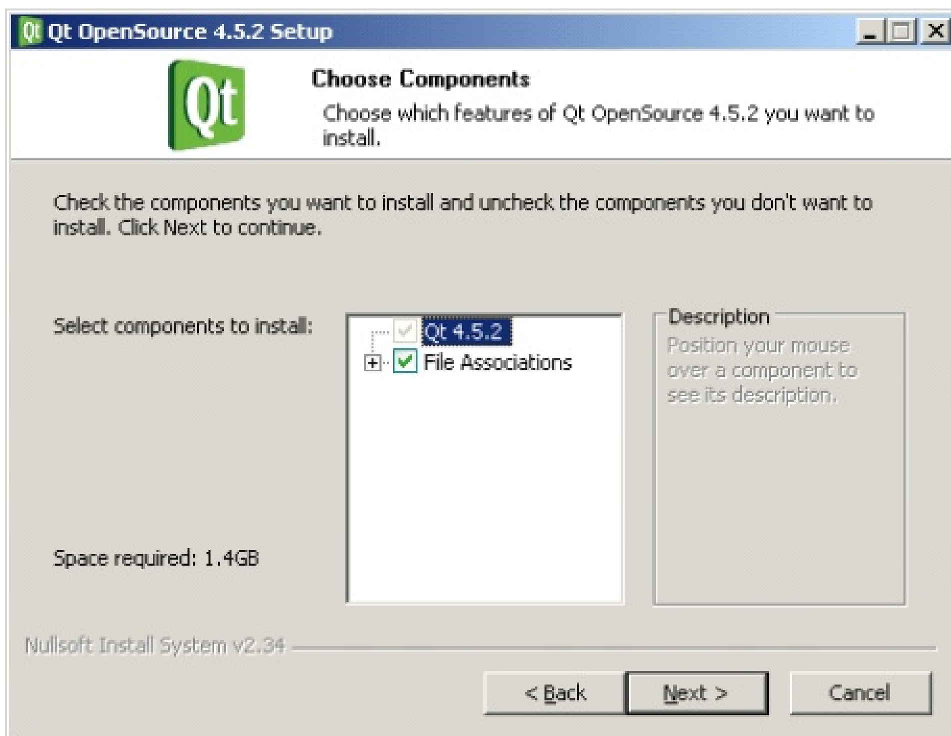


图 2-31 选择 Qt 组件

接下来的两步是选择 Qt 的安装路径和配置快捷方式，选择默认即可，再次提醒路径可 以自行决定，但不能包含空格和特殊字符。

下面这一步是安装过程中最为关键的。

如果你在前面还没有安装 MinGW，没有关系，在图 2-32 所示的画面中选择【Download and install minimal MinGW installation】，安装程序会先连接到 MinGW 的官方网站，下载并安装 MinGW。但根据笔者的体会，MinGW 的网站比较访问量比较大，下载速度较慢，所以种这方式相当耗费时间，一般光是这一步就要用去一个多小时的时间。

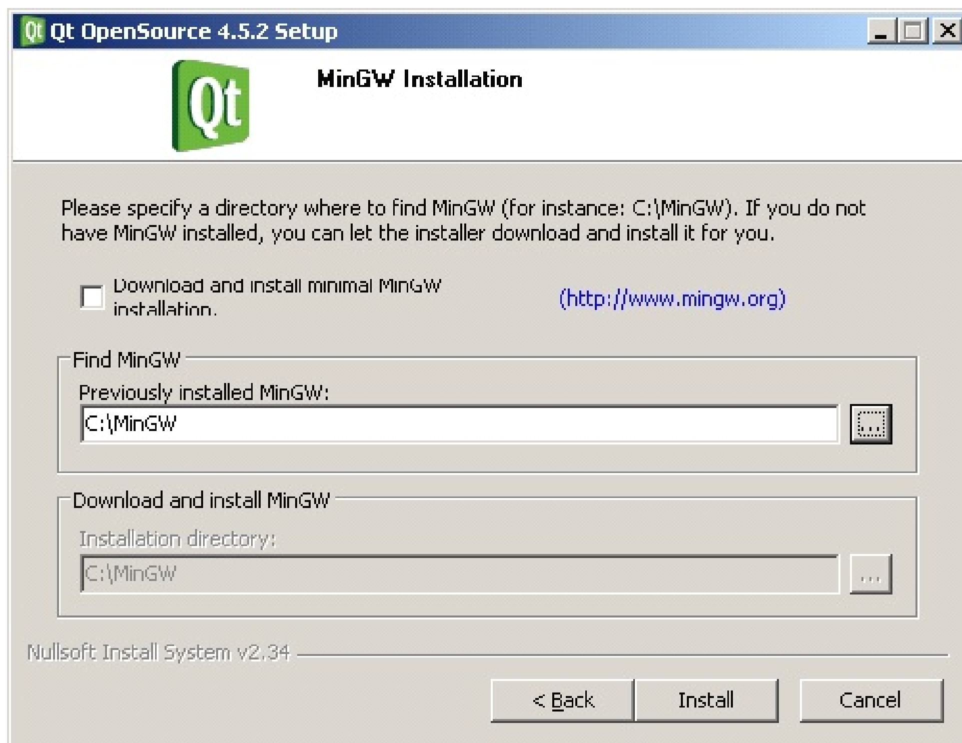


图 2-32 指定 MinGW 的位置（缺省选择）

如果你在前面已经安装了 MinGW，就不要选中【Download and install minimal MinGW installation】，而是在【Find MinGW】框中选择你安装的 MinGW 的实际路径。如果前面 安装 MinGW 时，你是用直接安装的方式，并采用默认的路径，这里你的选择就是如图 2-32 所示缺省的路径；如果你是采用安装 DevCpp 的方式安装 MinGW 的，就在其中选择为 Devcpp 的路径，如图 2-33 所示。

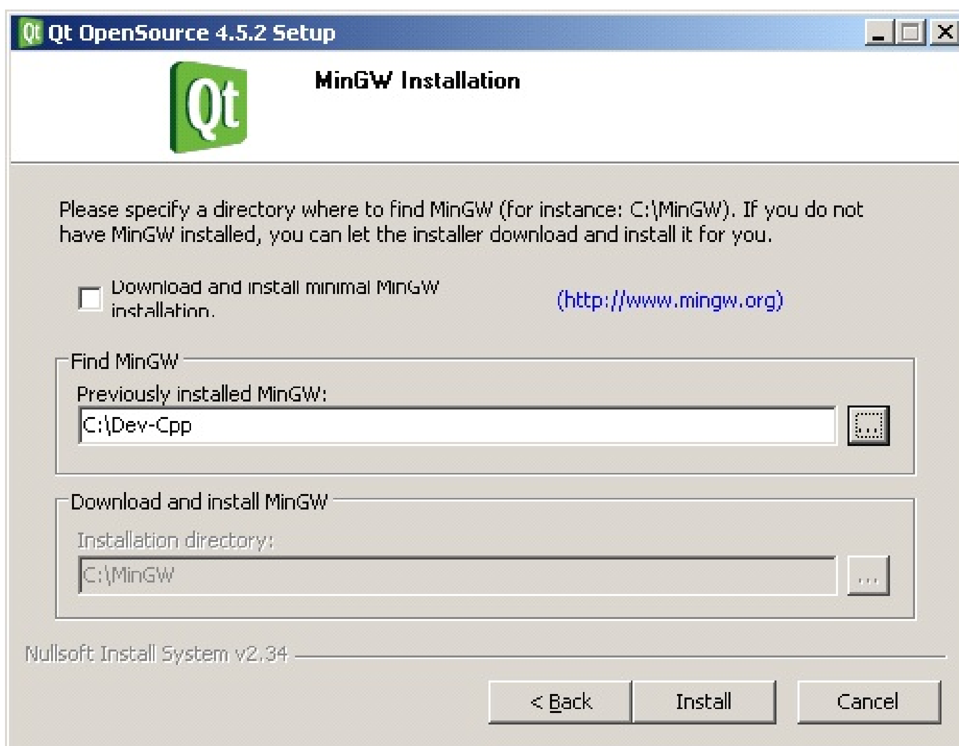


图 2-33 指定 MinGW 的位置（devcpp 方式）

后面的步骤就简单了，一路点击【Next】按钮直至安装结束，与 SDK 安装过程大致相 同，不再赘述。

(4) 后续工作

安装完成后，你将会在 Windows 的“开始”菜单中看到一个名为“Qt by Nokia(OpenSource)”的新的程序组。在这个程序组中，有指向的快捷方式，还有命令行提 示符（Qt4.5.2 Command Prompt）的快捷方式，它可以打开一个控制台窗口。打开这个窗口 后，它能够自动设置使用 MinGW 编译器编译 Qt 程序所需要的环境变量。在这个窗口中，可 以运行 qmake 和 make 命令来编译 Qt 应用程序。

(5) 配置环境变量 这与上一节中的步骤相同，把路径改为你安装的路径即可 。

4. 三安装方式的比较

在上面提到的 3 种安装方式中，采用 SDK 的方式无疑是最为便捷的，所需时间最少， 安装的组件最为全面，这也是官方重点推荐的方法，笔者也推荐。

采用源代码编译安装的方法是比较复杂的，尤其是在 Windows 上。而且这种方式并没 有安装 Qt Creator，它是“官方”推出的 IDE，也很好用。但这种方式能够使你更灵活的 配置你的 Qt 功能和版本，而且在这个过程中的苦辣酸甜都会使你对 Qt 的理解更进一步， 笔者也推荐这种方法。

采用框架安装的方式在难易度上介于前两者之间，安装过程算不上便捷，它也没有同 时安装 Qt Creator。此外，采用这种方法也相当耗费时间，光是下载和安装 MinGW 就要个 把小时，其中还要指定 MinGW 的位置，也可能出错。 比较起来，笔者不是很推荐采用这种方法。

所以，在 4.5 版以后，建议尽量采用 SDK 安装。随着学习的深入，并且需要灵活的配 置 Qt 库，那么就选择第二种的编译安装的方法。当然，如果你有时间，把这 3 种方式都熟 悉一下，是再好不过了，在这个过程中你会不断加深对 Qt 的理解，打牢你的基础，这对于 后来的学习过程有极大的裨益。

5.一点说明

Qt4.5 SDK 不能安装在 Windows 2000 平台上，这点在官方网站上并未说明，但事实确 实如此。

2.3.3 Qt/Mac 的安装

在把 Qt 安装到 Mac OS X 之前，必须已经安装了 Apple 的 Xcode Tools 工具包。这些 工具包通常会包含在和 Mac OS X 一起提供的那些 CD（或者 DVD）中，也可以从 Apple Developer Connection 中下载这些工具包，网址是 http://developer.apple.com。

如果使用的是 Mac OS X 10.4 和 Xcode Tools2.x（with GCC 4.0.x）或者是之后版 本，那么就可以使用如下描述的安装步骤；如果使用的是 Mac OS X 的早期版本，或者 GCC 是一个较早的版本，那么将需要手动安装这个源码包。该源码包的名称是 qt-mac- opensource-4.5.2.tar.gz，并且可以从 qtsoftware 的网站中下载它。如果要安装这个软 件包，就可以参照 X11 上编译安装 Qt 的方式，不要忘记配置 Qt 环境。

要使用安装程序安装 Qt，请下载 qt-mac-opensource-2009.03.1.dmg（在本书写作时 采用的就是这个版本，但当你阅读此书时，版本号可能已经与此有所不同了）。双击这 个.dmg 文件，然后再双击这个 Qt.dmg 软件包。这样将会启动安装程序，它会把 Qt 的文档 及其标准示例程序与 Qt 一起安装在/Developer 目录中。

要运行像 qmake 和 make 这样的命令，就需要使用一个终端窗口，例如，/Applications/Utilities 目录中的 Terminal.app。还有的情况就是要用 qmake 生成一个 Xcode 工程。例如，要为 hello 这个例子生成一个 Xcode 工程，就需要先打开一个控制台，比如 Terminal.app，然后把当前路径切换到 examples/chap01/hello，并且输入一下命令：

```
qmake -spec macx-xcode hello.pro
```

在 Mac OS X 上安装 Qt 的过程与 X11 上的情形很相似，只是目录有所不同，不再赘述。

2.3.4 Qt/WinCE

嵌入式系统中经常采用 WinCE 作为操作系统，因而在 WinCE 上安装 Qt 是很常见的，下 面就讲讲这方面需要注意的问题。本节将以一个实际的例子来说明整个的过程。 笔者的使用 的 Samsung i718 是基于 arm9 和 WinCE 的一款智能手机，如何在这个平台上面构建我需要的 Qt 开发环境呢。下面就是笔者的构思过程。

1.确定需求

了解需求这一步很重要，这可以使你少走弯路。比如要弄清楚，你在开发机上要使用什么操作系统，Windows 2000 还是 Windows XP，Vista；Qt4.5 是否支持它们，你的手持设备的操作系统是标准系统还是设备制造商自行定制的版本，开发平台是否匹配等。你一般还 需要在目标机和开发机之间传输文件和数据，这需要一个同步软件。

Qt 官方宣称从 4.5 版起，正式支持 Win CE，并已经在 Visual Studio 2005 上做过验 证，桌面操作系统（开发机）可以是 Windows XP 和 Windows Vista。Windows 2000 系列 并不在支持列表中，所以就不要选择了，至于选择那些古董级的 Windows Me、98 就更不现 实了，好在好像也很少有人这么做。

目标机系统可以是下列几种：

- Windows CE 5.0 for ARM, X86, SH-4 and MIPS
- Windows CE 6.0 for ARM generated using the defaults found in Platform Builder
- Windows Mobile 5.0 (Pocket PC, Smartphone and Pocket PC with Phone editions)

Windows Mobile 6.0 (Standard, Classic and Professional editions) 另外，你需要知道的是 Win CE 和 Windows Mobile 的关系，它们并不是等同的。 当然，你的手持设备制造商往往会自行定制一个基于上述系统的操作系统版本，你必须针对这些特点，有选择的调整编译参数，这样才能使 Qt 很好的支持开发。

我们回到实际的例子中。经过考虑，我决定开发机采用 Windows XP SP2 版，使用 Visual Studio 2005 中文版并打上 SP1 补丁，目标机是基于 Windows Mobile 5.0 Pocket PC 的，所以我们需要使用 Windows Mobile 5.0 Pocket PC SDK；另外，同步传输软件采用 Microsoft ActiveSync4.5 简体中文版，这可以在 Microsoft 的网站上下载到；Qt 库选用 qt-embedded-wince-opensource-src-4.5.2。表 2-1 归纳了我的需求分析的结果和最终的 系统选型。

表 2-1 需求分析及最终选型

需求	选型
CPU 架构	arm 系列（arm9）
开发机操作系统	Windows XP SP2 中文版
目标机操作系统	Windows Mobile 5.0 Pocket PC
开发机使用的 SDK	Windows Mobile 5.0 Pocket PC SDK
同步软件	Microsoft ActiveSync4.5 简体中文版
Qt 库	Qt/Win CE 4.5.2
开发 IDE	Visual Studio 2005 SP1 中文版

2.下载软件包

表 2-2 归纳了需要的软件包和下载地址。

表 2-2 软件包和下载地址

Windows Mobile 5.0 Pocket PC SDK	**http://www.microsoft.com**
Microsoft ActiveSync4.5 简体中文版	http://www.microsoft.com
Visual Studio 2005 SP1 中文版	购买
Qt/Win CE 4.5.2	http://www.qtsoftware.com

3.安装软件

按照以下步骤进行安装，次序不可弄错。

- 安装 VS2005，很简单，按照向导来，在自定义安装中务必选择 “智能设备开发”。
- 安装 Microsoft ActiveSync4.5 中文版，一路选择缺省配置即可；
- 安装 Windows Mobile 5.0 Pocket PC SDK，一路选择缺省设置即可；
- 安装 Qt/Win CE ， 下载到的文件名字类似于 qt-embedded-wince-opensource-src- 4.5.2

这里又分为几个步骤：

(1) 解压 将解压后的文件夹放到一个不含空格、中文字符和特殊字符的路径中，比如我的是：

```
d:\qt\qtWinCE，qtWinCE 即为此文件夹。
```

(2) 设置环境变量

鼠标右键单击“我的电脑”—>“属性”—>点“高级”标签—>“环境变量”选项—>

在"PATH"中添加路径“d:\qt\qtWinCE\ bin”。

(3) configure

进入到 VS2005 的命令行中，进入解压的文件夹，然后执行命令：

```
configure -platform win32-msvc2005 -xplatform wincewm50pocket-msvc2005
```

对于我的机器，也可以使用下面的命令，指出了具体的软硬件平台 ：

```
configure -platform win32-msvc2005 -xplatform wince50standard-armv4i-msvc2005
```

当然我们仍然可以运行 configure -help 命令来查看 configure 的参数选项，并根据自己的开发板及手机配置来选择其它的嵌入式的 xplatfrom，要了解这些信息，你可能需要 查看 Qt 所支持平台的 readme 文件里面的说明。configure 这个过程大约需要十几分钟。

(4) 更新环境变量

configure 正确完成后，为了使你的资源能够被目标机系统正确的找到，需要更新 Qt 环境变量，主要包括 PATH, INCLUDE 和 LIB 。命令如下：

```
set INCLUDE=C:\Program Files\Microsoft Visual Studio 8\VC\ce\include;C:\Program Files\Win
set LIB=C:\Program Files\Microsoft Visual Studio 8\VC\ce\lib\armv4i;C:\Program Files\Wind
set PATH=C:\Program Files\Microsoft Visual Studio 8\VC\ce\bin\x86_arm;%PATH%
```

(5) 运行 nmake

根据你机器配置的不同，大约需要 1 到若干个小时不等。nmake 执行成功后，你的 Qt for Win Ce 就可以使用了。

小贴士：一定要使用 VS 2005 打上 SP1 的的补丁。Qt/Win CE 的安装相对比较容易，但是更新环境变量那一步容易被忽略。

2.3.5 Qt/S60

自 4.5 版后，Qt 将支持 S60，并且可以与 Qt Creator 集成。在笔者写作时，最新的版 本是 Qt for S60 的技术概览版（Technology Preview）Tower，这已经是第 2 个概览版 了，Qt for S60 完全版计划于 2009 年第四季度发布。虽说不是正式版，但它的安装方式 和特性与正式版并无太大差异。但请注意，技术概览版包含尚不成熟的代码，还未达到最终 发布产品的性能和兼容性。

1.了解需求

表 2-3 归纳了在 S60 上安装 Qt 所需的软件包名称以及它们的最低版本和需要的环境。

表 2-3 需要的软件包

软件名称	版本	说明
ActivePerl	5.6.1 或者更高	执行脚本文件环境
JRE	JRE 1.5 或更高	Java 环境
S60 SDK	S60 SDK 3rd FP2 for C++ 或更高（依硬件不同而不同）	Series 60 软件开发环境
C++编译器	Carbide.c++ V2.0、V C6.0 或更高（Qt4.5 不再支持 VC 6.0）	Carbide.c++、Visual C++等
Qt for S60 source code	4.5.2 或更高	Samsung 部分手机有专门版本
操作系统	Windows XP SP2 或更高	

2.准备安装

如果安装过程中有意外，对应的帮助可以在 C:\qts60\doc \html\s60-with-qt- introduction.html 里找到。

表 2-4 归纳了我举例时选用的软件包和版本，还有获取它们的网址。

表 2-4 选用的软件包的详细情况

软件包	选用版本	下载网址
ActivePerl	5.8.8	http://www.activestate.com
JRE	JDK7.0（含 JRE）	http://cn.sun.com
S60 SDK	S60 SDK 3rd FP2 for C++	http://www.forum.nokia.com
Carbide.C++	V2.0	http://www.forum.nokia.com
Qt for S60 source code	4.5.2 Technology Preview Tower	Http://www.qtsoftware.com
Windows	Windows XP SP2 中文版	购买

小贴士：在安装 Qt for S60 之前，我们必须先安装配置好 S60 的开发环境。首先必须知道 的一件事情是，所有的与 S60 相关的开发工具，最好都安装在同一个逻辑盘里面，并且在 安装的路径中不要包含有空格，比如， "Program Files"，否则，可能安装不成功，或者在 后面的开发中会遇到莫明其妙的问题。我把所有这些软件全部安装到了 d:\QtS60 目录下 面。

再有就是安装这几个软件是有顺序的，其中最为重要的是在编译安装 Qt for S60 之 前，最后一个安装 Carbide.C++ ,否则，即使最后安装成功，却也可能不可以建立 Qt 工 程，切记！我选择的顺序是 ActivePerl、JDK、S60 SDK、Carbide.C++，最后是 Qt。

下面是详细的安装步骤。

(1) 安装 ActivePerl 这个也没有什么好说的了，基本是一路 点击【Next】按钮，其间只需要更改安装路径。我的情形如图 2-34 所示。

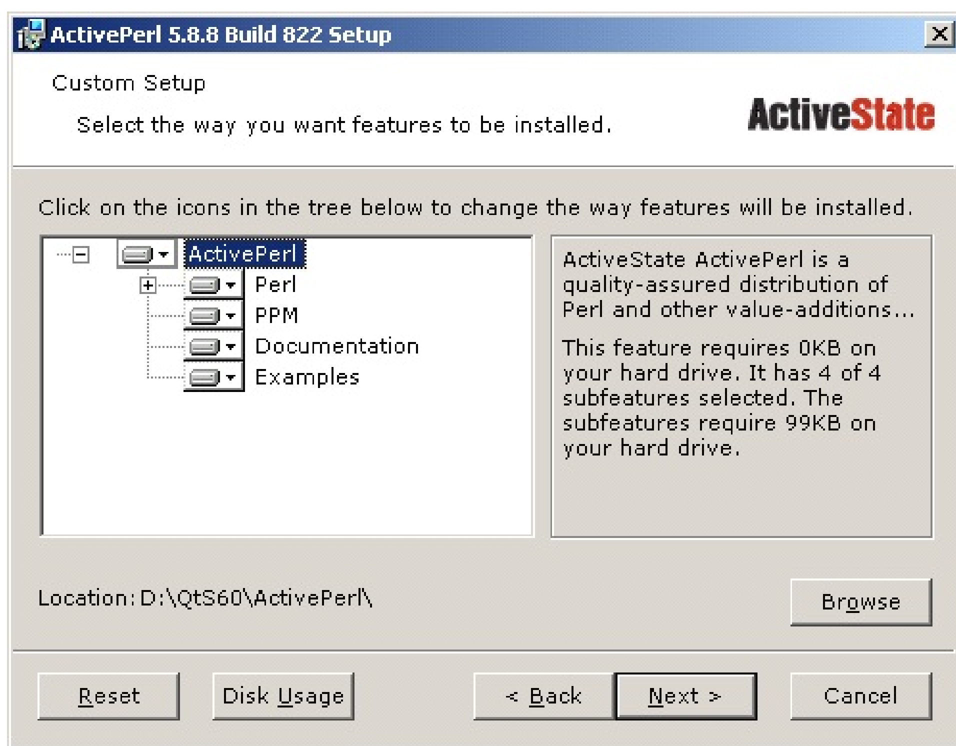


图 2-34 ActivePerl 安装路径

(2) 安装 JRE

我是安装的 JDK，中间过程需要关注 JDK 和 JRE 的路径。 安装 JDK 的路径如图 2-35 所示。

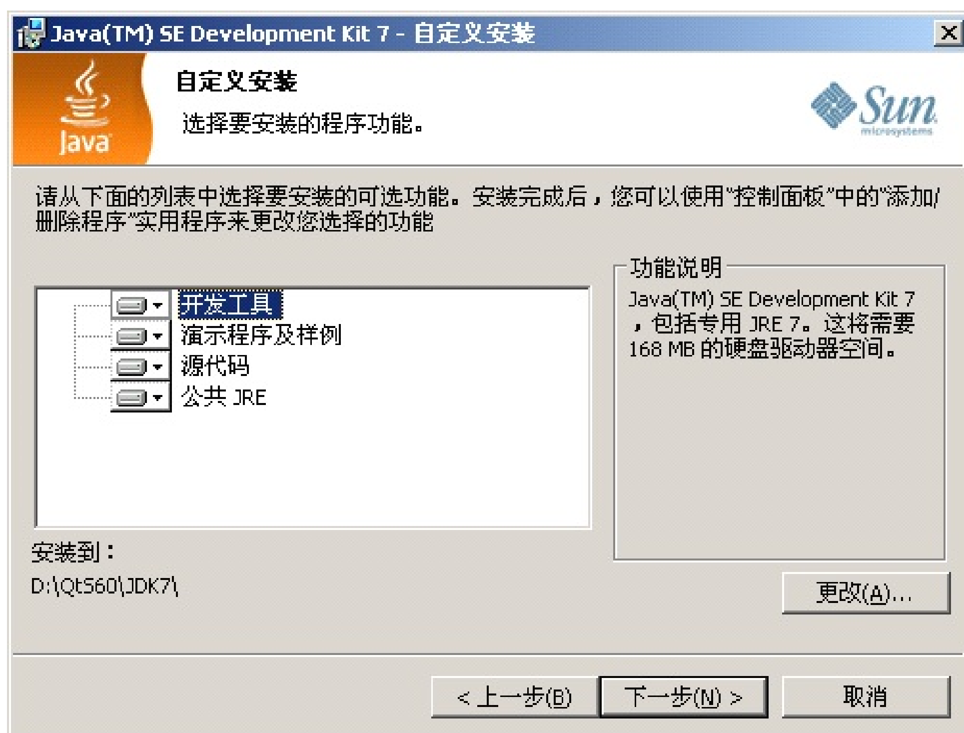
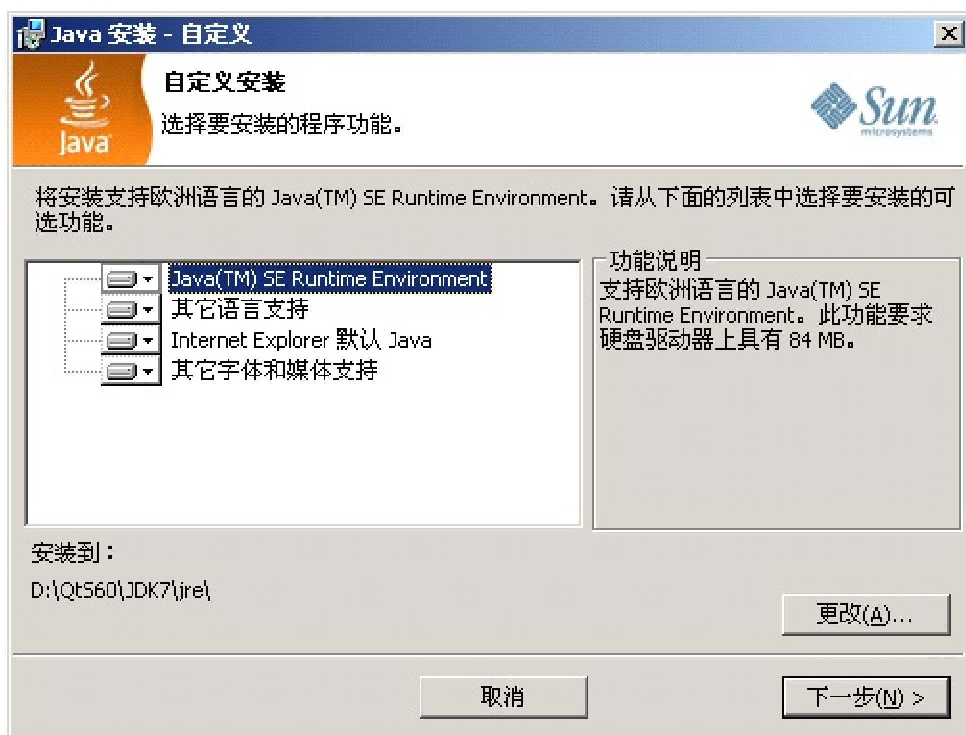


图 2-35 JDK 的路径

安装 JRE 的路径如图 2-36 所示。



(3) 安装 S60 SDK 3rd FP2

图 2-36 JRE 的路径

这一步有几个地方需要注意，第一个是路径，我设置的路径如图 2-37 所示。

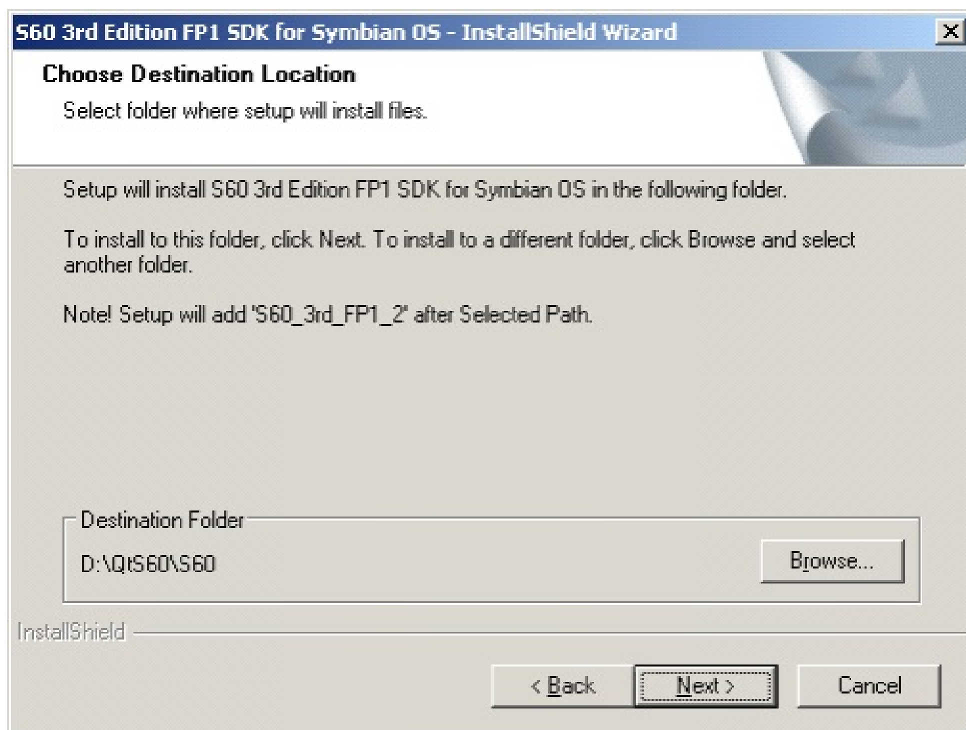


图 2-37 S60SDK 的路径

第 2 个地方是选择一下版本，我的情形如图 2-38 所示，这里只有一个选项，但必须选上才行。

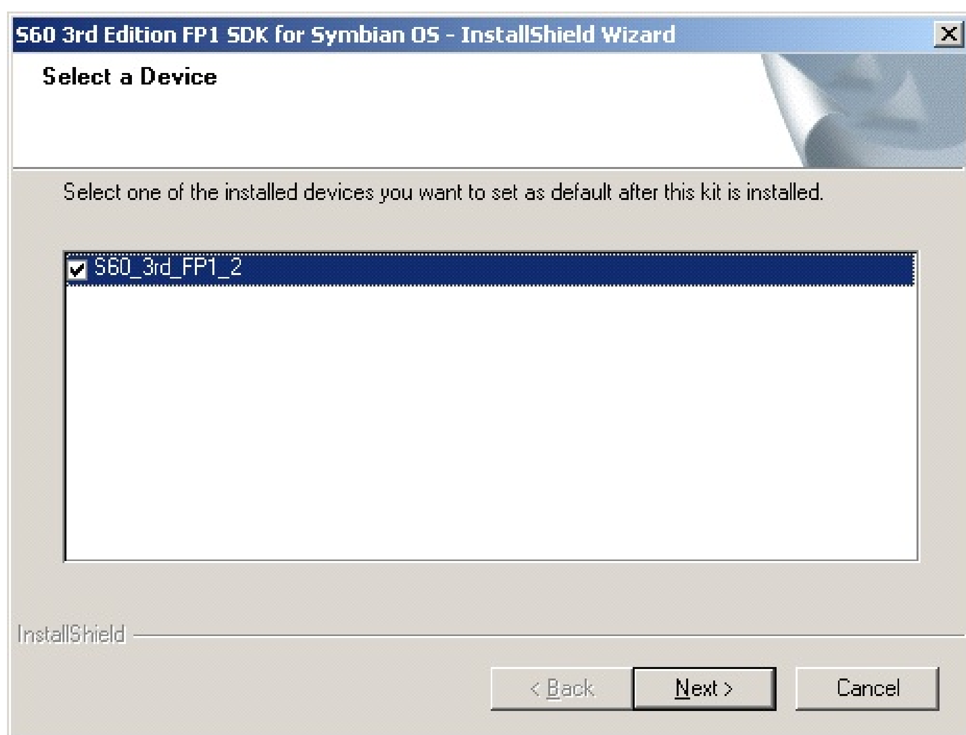


图 2-38 选择安装组件

然后都是选择缺省设置，一路点击【Next】按钮，就可以完成安装。 安装完成之后，验证一下是否可以成功启动 S60 的模拟器，如果启动不了，说明安装不成功。运行模拟器可以依次点击： Windows Start Menu | Programs | S60 Developer Tools | 3rd Edition FP2 SDK | v1.1 | Emulator, 这个过程如图 2-39 所示。

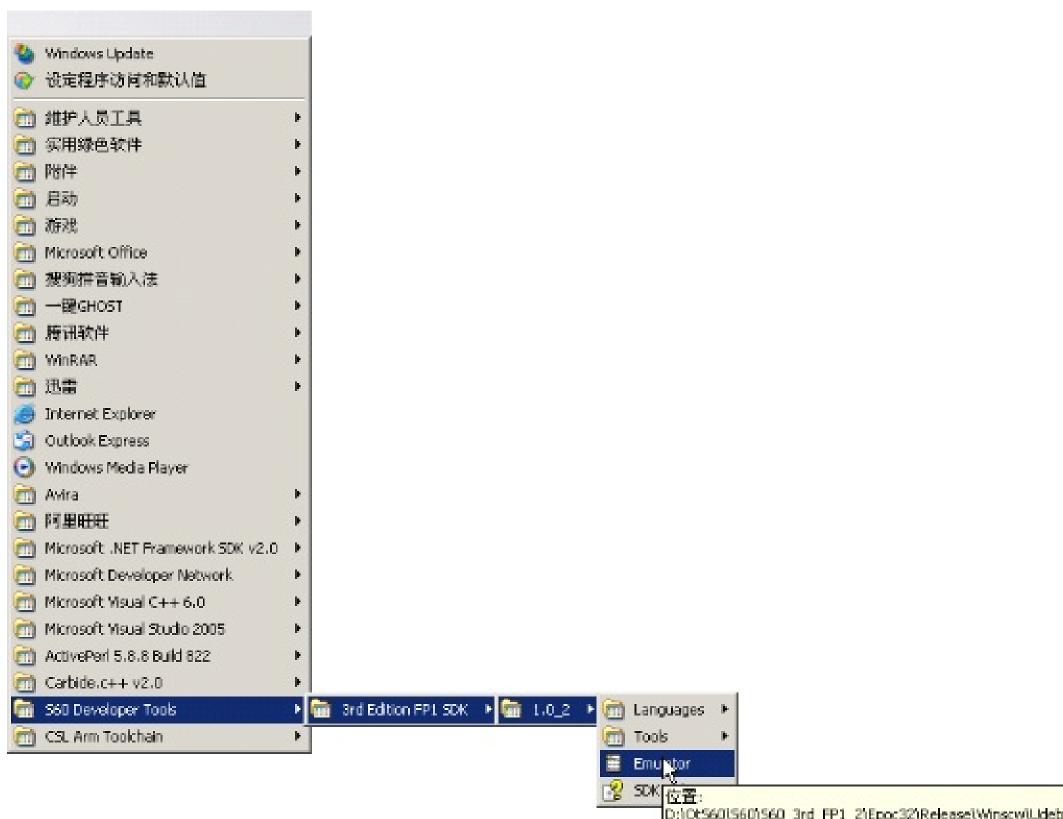
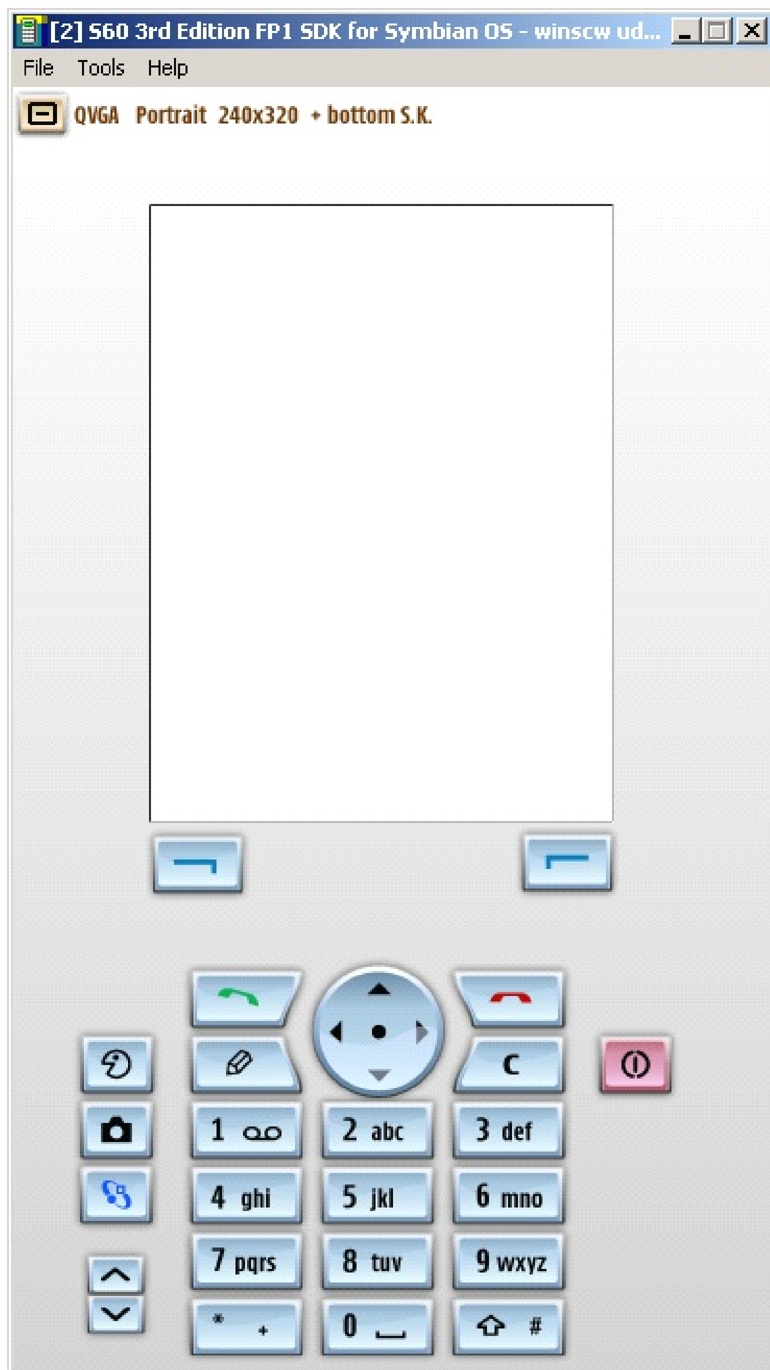


图 2-39 启动 S60 的模拟器

启动后的模拟器样子如图 2-40 所示，可以点选上面的手机按钮测试一下。



(4) 安装 Carbide.c++

图 2-40 S60 模拟器

这里有几个地方需要注意，一个是选择安装的版本， 如图 2-41 所示，这里选择 Professional Edition，它的功能最全。

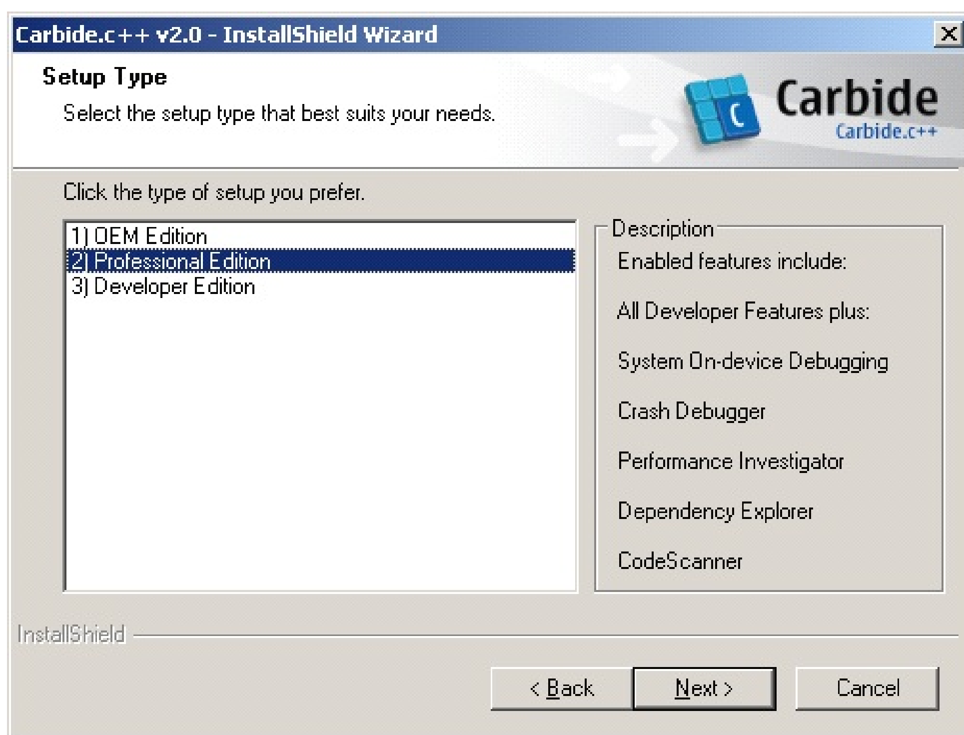


图 2-41 选择 Carbide.C++的版本

还有就是一直强调的路径问题，其情形如图 2-42 所示。

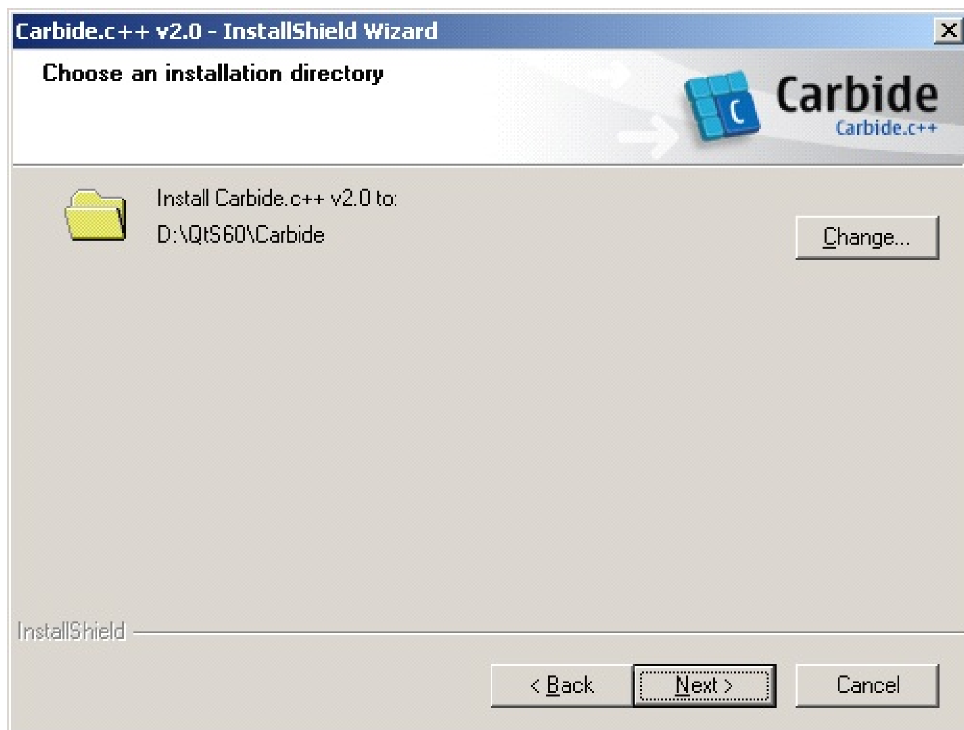


图 2-42 Carbide.c++的安装路径

选择好安装路径后，一路点击【Next】按钮，直至安装结束。这时，安装程序会如图2-43 所示那样提醒你如果日后需要使用命令行工具，就需要配置 WINSCW 环境变量，如果完 全使用 IDE，则不必进行这一步。



图 2-43 安装的提示

我是觉得有必要，按照提示的说法，编译一下 WINSCW 的环境变量，大约需要十几秒的 时间就好了，如图 2-44 所示。

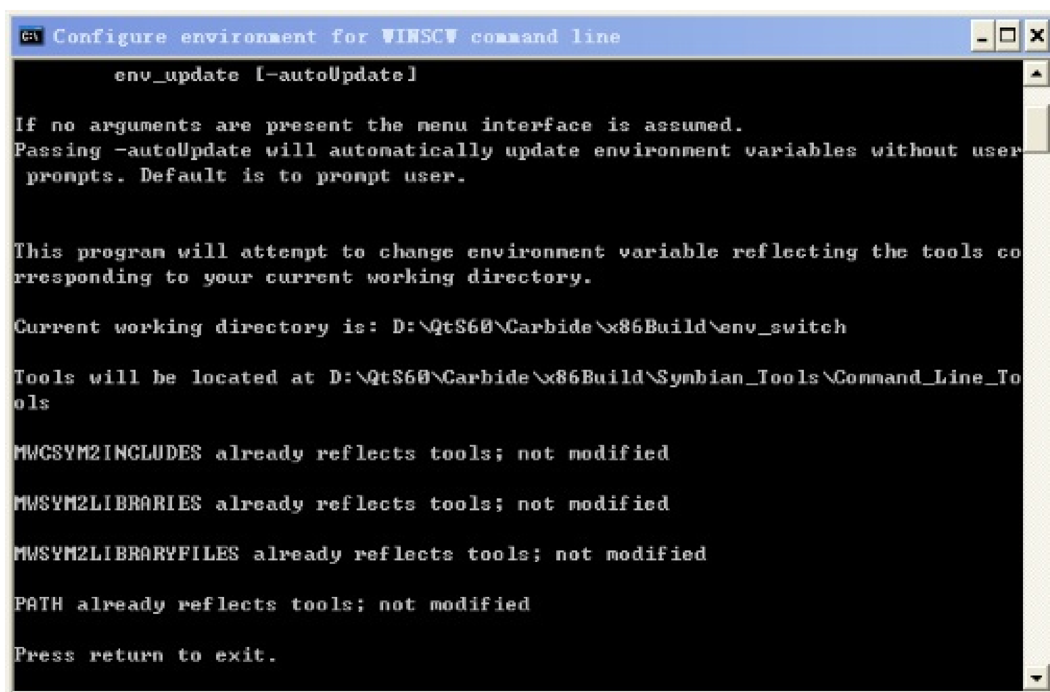


图 2-44 编译 WINSCW 环境

安装完成之后，启动 Carbide.c++，第一次运行 Carbide.C++，它会要求设置 workspace 的路径，这个也很重要，这个路径设置一定要和 S60 SDK 在同一个逻辑盘上，不 然，后面编译程序的时候会出错，如图 2-45 所示。

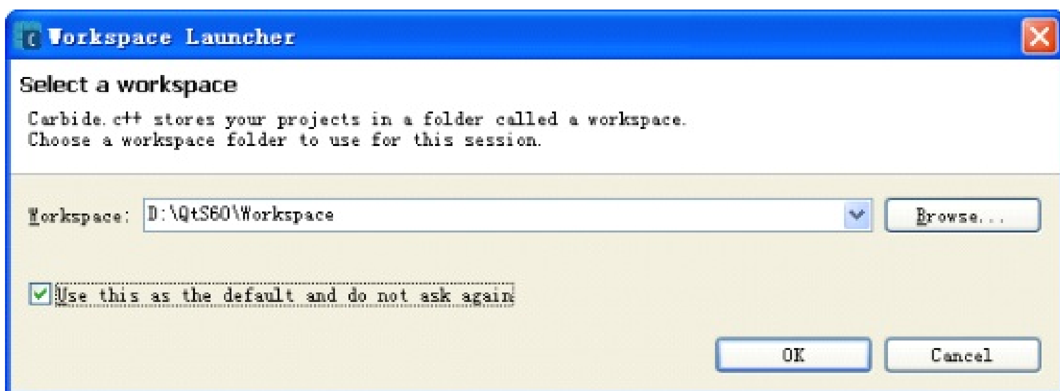


图 2-45 设置 Workspace

在启动之后，它会自动去扫描系统中的 S60 SDK，然后给出一个提示，需要重启 Carbide.c++，满足它的要求，点击 Restart 按钮重新启动即可，如图 2-46 所示。



(5) 安装 Qt for S60

图 2-46 Carbide.c++要求重新启动

首先是解压缩 Qt for S60 源代码包，把它放到与前述软件相同的逻辑盘中的一个路径 里面，这里是 d:\QtS60\QtS60SDK，路径可根据自己情况调整。

然后是配置环境变量，把 Qt for S60 的 bin 子目录路径加入到 PATH 中去，目的是要 在后面 configure 时找到 qmake 等工具。例如我的就是 d:\QtS60\QtS60SDK，可以通过命令 行或者【Control Panel】->【System】->【Advanced】->【Environment variables】来完 成。

接下来开始配置 Qt。打开一个一个命令行界面的窗口，切换到解压后的 Qt for S60 软 件包的目录，我的是 d:\QtS60\QtS60SDK，执行命令：configure -platform win32-mwc - xplatform symbian-abld 开始 configure。如图 2-47 情形，系统会首先询问要安装哪个版 本，商业版（commercial）还是开源版（Open Source），我们输入 o，选择开源版。

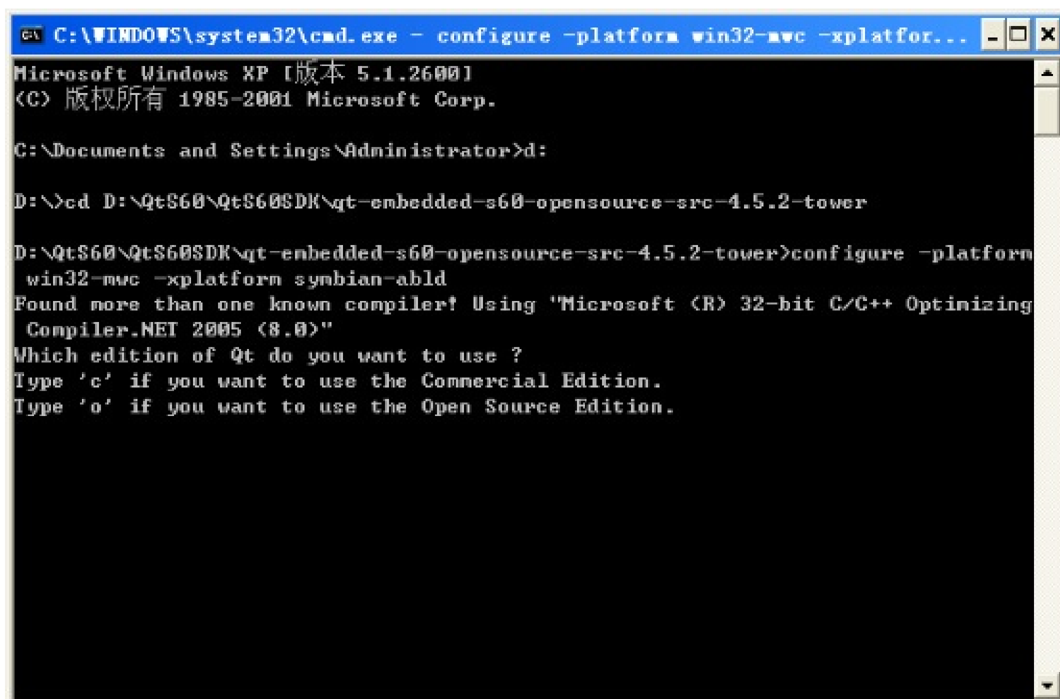


图 2-47 configure 的情形

这之后，如图 2-48 所示，Qt 会询问是否同意它的 License，当然接受，输入 y,按下回 车，开始配置。

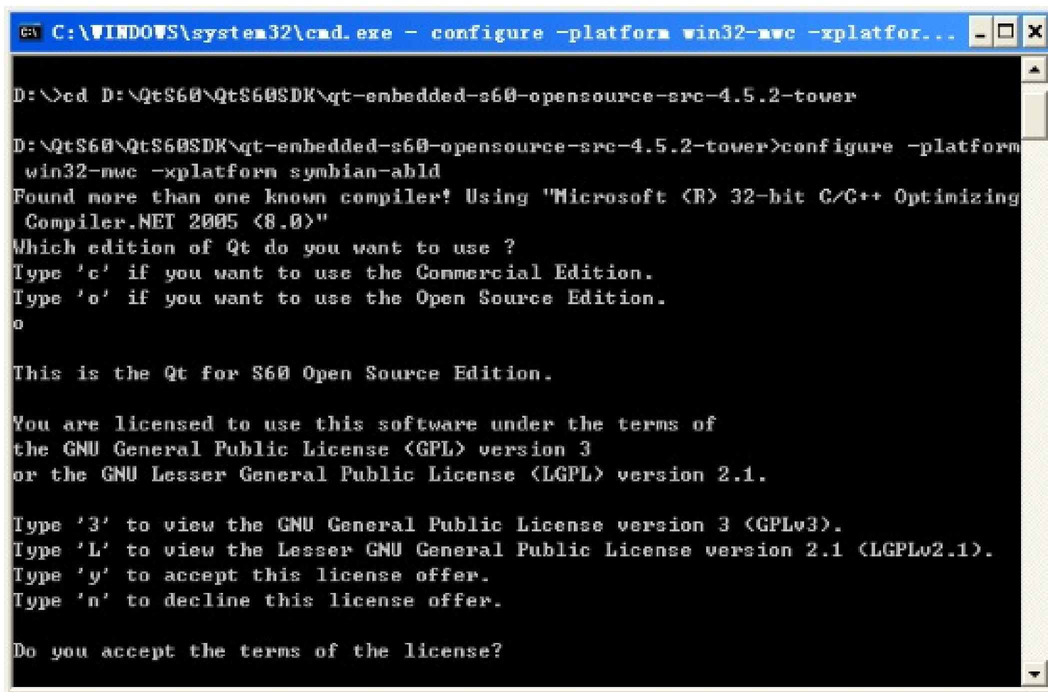


图 2-48 同意 Qt 授权

经过一段时间，configure 结束，我们开始编译 Qt。输入命令：

```
make debug-winscw
```

经过若干小时，编译结束，Qt 算是安装完成。注意，这里同样不需要 make install。接下来，我们需要配置一下 Carbide.c++，使它能够与 Qt 很好的集成。如图 2-49 所示，启动 Carbide.c++。

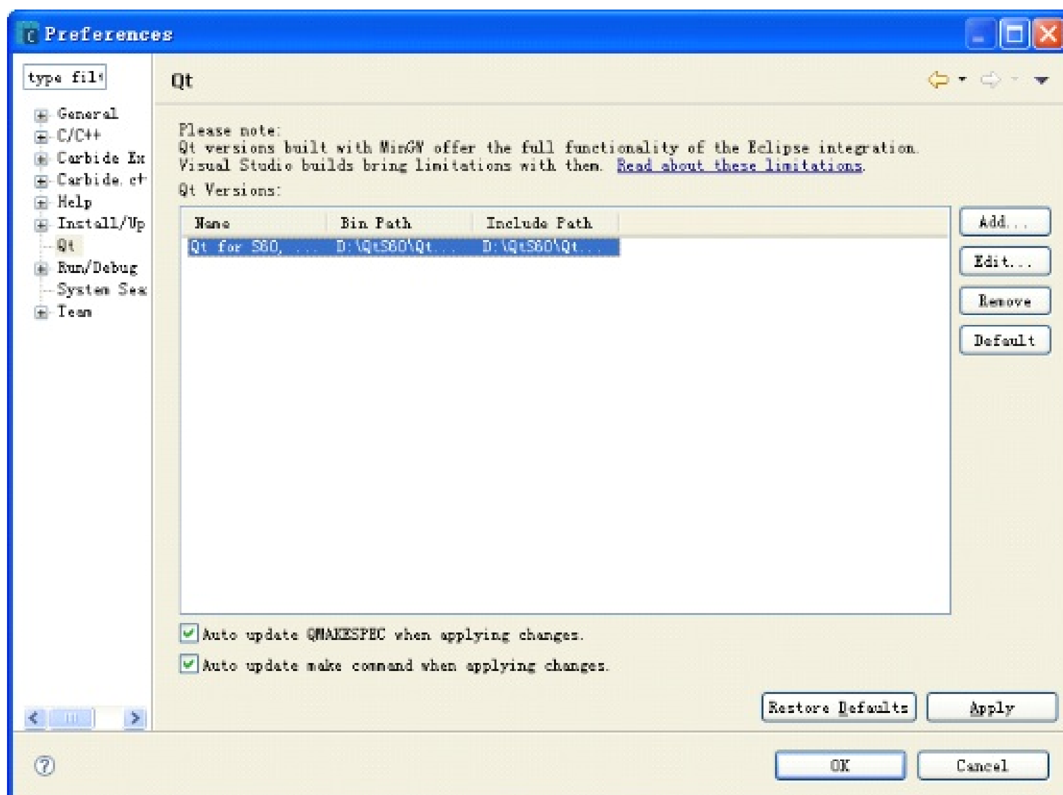


图 2-49 设置 Qt 的 Preference

然后依次点击主菜单的 Windows | Preference，在左边的列表中选择 Qt，然后点击右 边的【Add...】按钮，弹出对话框，如图 2-50 所示，添加添加 Qt 的目录，Version Name 一 项可以随意命名。

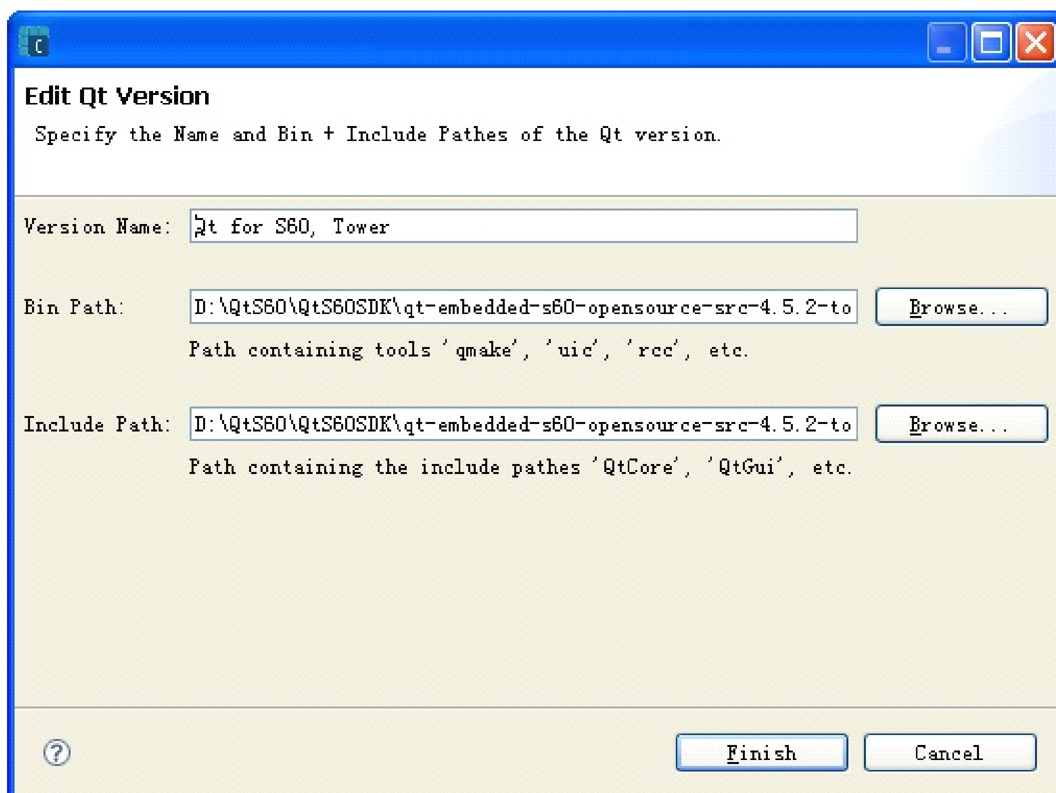


图 2-50 配置 Qt 的路径

好了，到这里 Qt for S60 开发环境就搭建完成了，在这个平台上的工作将是使用 Carbide.c++结合 Qt 进行来完成。

2.3.6 基于 Linux 发行版的安装

1.Red Hat 9.0

RedHat 9.0 是一个比较“古老”但直到现在还有很多人在使用的版本，在论坛上不时 的可以看到有朋友提问与安装 Qt 相关的问题。所以，笔者觉得有必要将 Red Hat 9.0 上安 装 Qt 的方法详细介绍给大家。

RedHat 9.0 上自带的 Qt 版本是 Qt3.1.1 的，并且 Red Hat 已经不再对它进行升级和支 持。所以一般只能采用编译源代码的方式来安装 Qt4。

(1) 下载 Qt4 源码包

先去官方网站下载 Qt 的源码包，版本为“qt-x11-opensource-src-4.5.2.tar.gz",下 载到 linux 中解压。

(2) 修改头文件链接 打开个终端，输入：

```
ln -s /usr/kerberos/include/com_err.h /usr/include/
ln -s /usr/kerberos/include/profile.h /usr/include/
ln -s /usr/kerberos/include/krb5.h /usr/include/
```

这样设置的目的是防止在下面的编译中，报出一个常见的 “krb5.h”的错误。

(3) 修改头文件内容

有时候，在下面进行的编译安装时，会报出 “TIFFReadRGBAImageOriented' undeclared”的错误，这需修改里面的一些文件。

修改方法是：打开 qtiffhandler.cpp 把
`TIFFReadRGBAImageOriented(tiff, width, height, reinterpret_cast<uint32 *>(tiffImage.`
改 为
`TIFFReadRGBAImage(tiff, width, height, reinterpret_cast<uint32*>(tiffImage.bits()),`
就可以了，因为 RedHat 9 里的 usr/include/tiffio.h 没有前 一个函数。

(4) 编译安装 Qt

下面开始开始编译安装 Qt。

在终端内，cd 到解压出来的文件下面，然后输入：

```
./configure
gmake
gmake install
```

整个过程大约需要若干个小时，视你的机器速度不同而有长短。它安装的默认路径是：/usr/local/Trolltech/Qt-4.5.2。

(5) 配置 Qt4 的环境变量

I. 打开 /etc/profile 文件，在该文件的末尾加上以下语句：

```
PATH=/usr/local/Trolltech/Qt-4.5.2/bin:$PATH
QTDIR=/usr/local/Trolltech/Qt-4.5.2
MANPATH=$QTDIR/man:$MANPATH
LD_LIBRARY_PATH=$QTDIR/lib:$LD_LIBRARY_PATH
export PATH QTDIR MANPATH LD_LIBRARY_PATH
```

II. 重定向链接

系统默认开启的是 Qt3，现在我们来修改，启动 Qt3 的关联文件在“/usr /bin”下 面，找到“designer”，我们发现这个文件果然链接的是 Qt3 的执行文件，那么就清楚了 designer 是个链接，通过 designer 指向 Qt3 的 designer, 现在想让 designer 默认为 qt4 的 designer, 只要重定向链接。同样，想要 qmake 也默认为 Qt4 的也一样，只要重定向链 接。以下是方法：

```
rm /usr/bin/designer
ln -s /usr/local/Trolltech/Qt-4.5.2/bin/designer /usr/bin/designer
rm /usr/bin/qmake
ln -s /usr/local/Trolltech/Qt-4.5.2/bin/qmake /usr/bin/qmake
```

我的建议是最好把那个跟“/usr/local/Trolltech/Qt-4.5.2”下面的执行文件有关的

都修改一下链接。这样便可以彻底的使用 Qt4 了。

(6) 安装修改字体

这里使用 Qt4 默认的字体会出现乱码，将系统采用的字体设置为中文字体，我们来设 置一下。打开 qtconfig，如图 2-51 所示，将<Select or Enter a Family>和<Select Subtitute Family>都设置为 Bitstream Charter，然后保存退出即可。

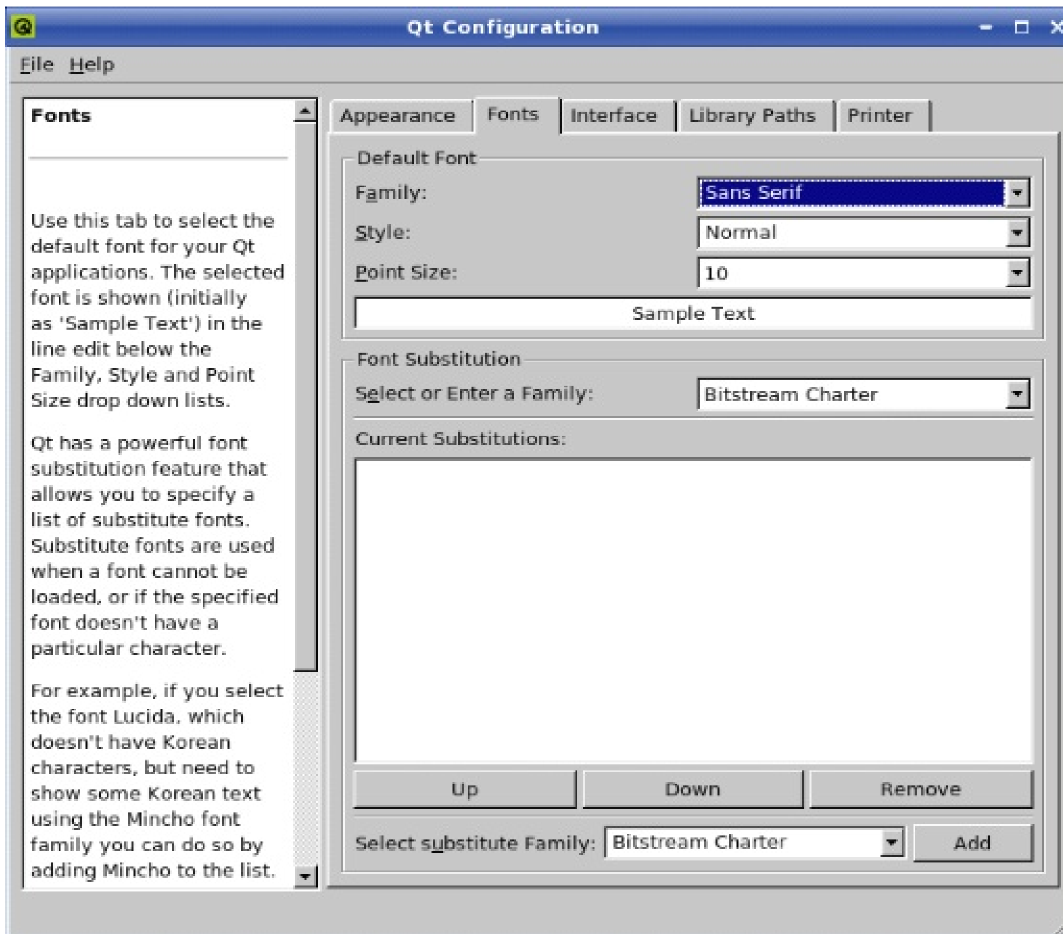


图 2-51 在 Qt Config 中配置字体

这样，我们的 Qt4 就可以在 Red Hat 9.0 上使用了。

2.Fedora Core

这里我们以 Fedora Core 10（以下简称 FC 10）为例，介绍在 Fedora 上如何安装配置 Qt4 的开发环境。

在 FC 10 上安装配置 Qt4，对于新手而言，我们一般不推荐采用编译源代码的方式。 由 于在默认情况下，FC 10 采用 GNOME 环境，而 Qt4 又需要依赖与 KDE 相关的好多库，并且 FC10 的软件包管理器在处理依赖问题时不是很方便，所以 编译安装 Qt4 容易出错且不好解决。因此我们推荐采用 FC 10 编译好的安装包来安装 Qt4，一般要遵循下面的步骤：

(1) 安装系统

FC 10 的安装比较简便，建议最好采用 Live CD 的方式来安装，按照向导的提示，通常 只需若干个步骤即可完成。

(2) 更新系统

安装完毕后，需要做的第一件事就是更新系统。步骤如下： 依次点击菜单栏上的【系统】->【管理】->【更新系统】，出现如图 2-52 所示的对话框，系统将检测你需要更新的内容，待系统检测完毕后，点击【Update System】按钮即可开始更新，需要的时间与你机器的速

度和网络带宽有关系，大概从几十分钟到若干小时不 等。



图 2-52 更新系统

(3) 安装 Qt

完成系统更新后，我们开始安装 Qt。依次点击【菜单栏】->【系统】->【管理】->【增加/删除软件】,系统弹出软件包管理器的界面，如图 2-53 所示。

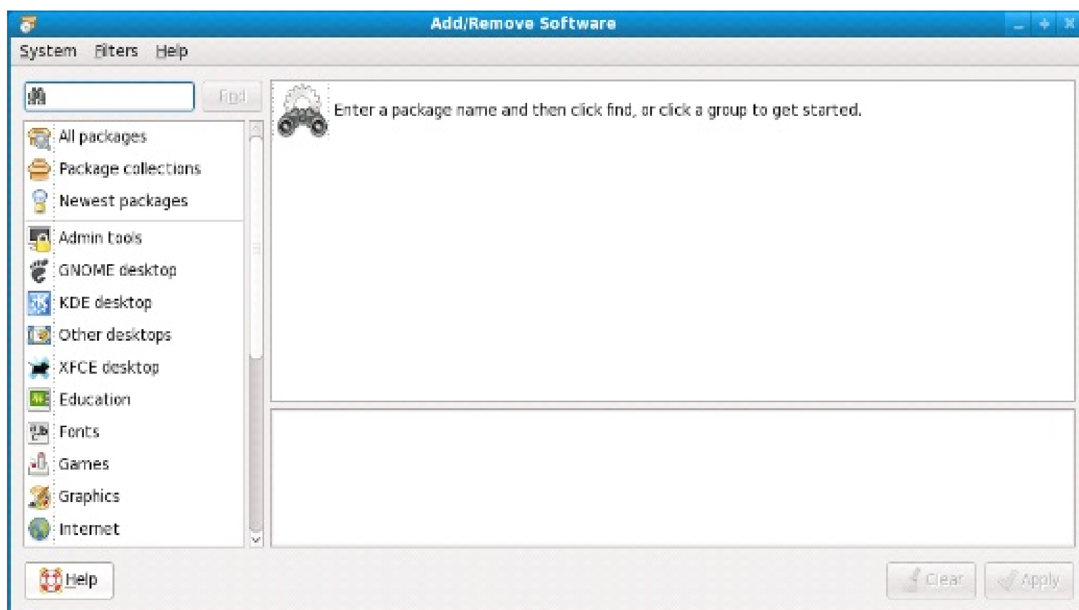


图 2-53 FC 10 的软件包管理器

I. 配置过滤条件

如图 2-54 所示，点击【Filters】，在下拉的列表中取消选中默认的【 Only newest packages】。

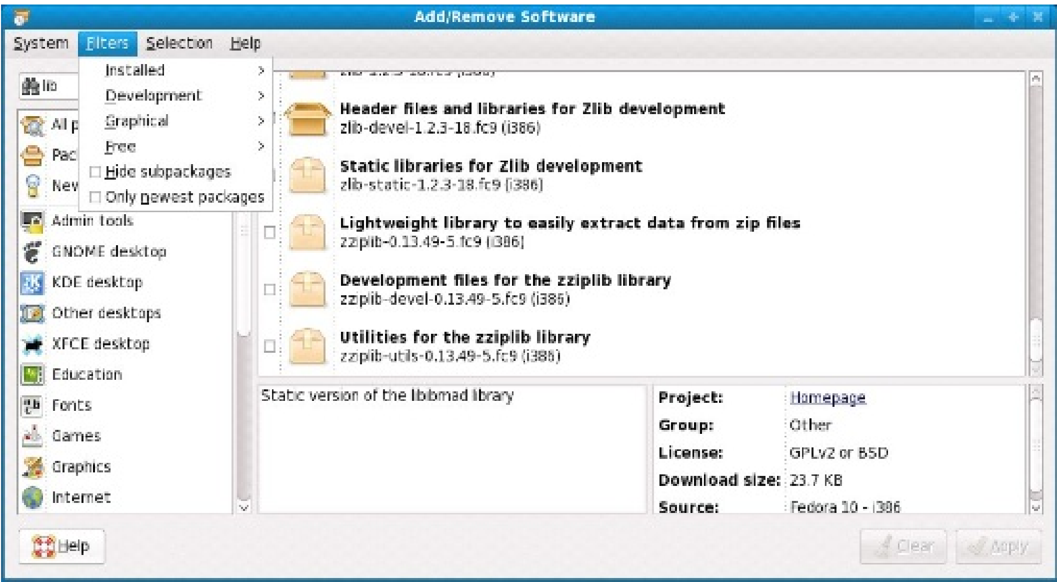


图 2-54 配置过滤条件第 1 步

然后，在每个二级菜单下面，选中【No filter】,如图 2-55 所示。

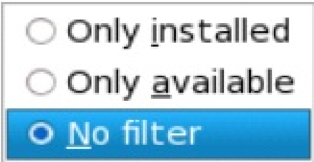


图 2-55 配置过滤条件第 2 步

II. 安装 Qt

如图 2-56 所示，在搜索栏中输入“qt”，然后按下回车键或者鼠标点击【Find】按 钮，软件包管理器将把所有包含 qt 关键字的包罗列出来，表 2-5 列举了我们要安装的包。

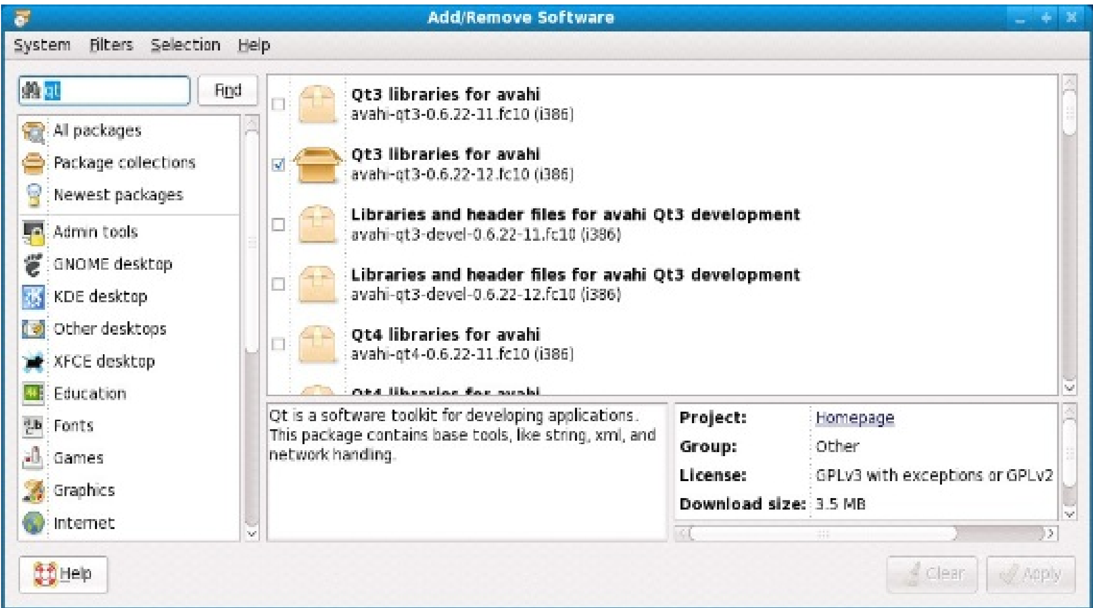


图 2-56 搜索 Qt 的安装包

表 2-5 需要安装的软件包

包名称	说 明
qt-x11-1:4.5.0-14.fc10(i386)	Qt 基础库，包括 Qt Designer
qt-1:4.5.0-14.fc10(i386)	Qt 基础工具，含 xml、sql、network 等模块
qt-devel-1:4.5.0-14.fc10(i386)	Qt 的开发文档，包括 Qt Linguist
qt-doc-1:4.5.0-14.fc10(i386)	API 文档、帮助和示例，包括 Qt Assistant 和 Qt Demo
qt3-config-3.3.8b-17.fc10(i386)	Qt 配置工具
scim-bridge-qt-0.4.15-8.fc10(i386)	为 Qt 配置输入法支持
scim-qtimm-0.9.4-11.fc10(i386)	为 Qt 配置输入法支持
qt-creator-1.1.0-2.fc10(i386)	安装 Qt Creator，可选
qt-mysql-1:4.5.0-14.fc10(i386)	安装 MySQL 数据库驱动
qt-odbc-1:4.5.0-14.fc10(i386)	安装 odbc 驱动
qt-postgresql-1:4.5.0-14.fc10(i386)	安装 PostgreSQL 数据库驱动

(4) 配置环境

请参见后面的《配置环境》一节。这里需要说明的是，在某些发行版中，由于自行增 加了某些配置文件，与标准的配置文件可能有些不同，但 FC 属于“遵守规定”的。也就是 采用标准配置方法就可以了。

至此，在 FC 10 上安装配置 Qt4 就完成了，这之后可以在程序组中找到 Qt4 各个组件

的快捷方式，并且使用它们开发应用程序了。

3. Ubuntu/Kubuntu

Ubuntu 和 Kubuntu 这两个 Linux 发行版都是比较容易上手的，在上面安装 Qt4 也是比 较简捷的。但在论坛里面还是不断有朋友提问与 Ubuntu 上安装配置 Qt4 相关的问题。下面 笔者就以 Ubuntu8.04 为例，讲解从 Live CD 安装 Ubuntu 后如何安装配置 Qt4 的全过程， 希望能够对读者朋友有所帮助。

(1) Ubuntu 版本的选择

截至笔者写作时，Ubuntu 最新的版本是 9.04 版，它带来了性能上的提升与多方面的改 进。但是笔者仍然向大家推荐 8.04 版，原因是这是 Ubuntu 最近几年来推出的一个 LTS 版 (长期支持版)，并且技术比较先进，软件仓库中提供的软件能够持续更新，服务更有保障。 上一个 LTS 版是 6.06 版，技术已经比较落后了，下一个 LTS 版还没有浮出水面。

(2) 安装方式的选择

Ubuntu 8.04 支持多种安装方式，这里推荐采用 Live CD 安装的方式。Live CD 是免费 的，可以在 Ubuntu 的官网上提出申请，Ubuntu 会免费邮寄给用户。也可以在下载安装文件 后，刻成 Live CD 来使用，可以登录 Ubuntu 中文官方论坛查询有关信息。

(3) 添加软件源

在源里面添加两个地址：

```
deb http://download.tuxfamily.org/3v1deb feisty eyecandy
deb-src http://download.tuxfamily.org/3v1deb feisty eyecandy
```

Ubuntu 的源文件在/etc/apt/sources.list 里，将上述两行添加入来。

(4) 安装 Ubuntu

采用 Live CD 方式安装 Ubuntu 只需要 6 步即可完成，只需依据安装向导的提示完成安 装即可。

(5) 执行系统更新

I. 安装完系统后，依次点击菜单【系统】->【系统管理】->【更新管理器】，系统弹 出如图 2-57 所示的界面，首先点击【检查】按钮，系统将会检查需要更新的内容，几分钟 后，系统将会提示你需要更新的内容和大约的时间 。



图 2-57 更新管理器

II. 点击【安装更新】按钮，系统将会执行 Update 操作，时间视你的网络速度和需要 更新内容的多少而又不同。需要强调的是，你的机器需要保持互联网在线。

(6) 中文化

I. 执行完系统更新后，依次点击【系统】 -> 【系统管理】-> 【语言支持】， 如果是第 一次运行，系统会提示说，语言支持选项尚未安装完全。则需要依据提示将汉语（中国）以 及其它你想安装的语言安装上去，如图 2-58 所示。



图 2-58 安装语言支持

II. 在新立得软件包管理器中搜索：language 关键字，将 language-pack-gnome-zh、language-pack-gnome-zh-base、language-pack-kde-zh、language-pack-kde-zh-base、language-support-extra-zh、language-support-fonts-zh、language-support-input- zh、language-support-translations-zh、language-support-zh 这些包选中并安装。

III. 安装中文字体，比如选中 ttf-wqy-zenhei 包即可安装上“文泉驿正黑”这种中文 字体。

(7) 安装输入法

在新立得软件包管理器内搜索 scim 关键字，将 scim、scim-chewing、scim-chinese、 scim-pinyin 包选中并安装。

(8) 闪电配置系统环境 这一步主要是安装配置系统的字体、程序常用的插件、常用多媒体编码插件，以及应用环境等。只需在新利得软件包管理器中搜索并安装 ”ubuntu-restricted-extras”软件包即可，一次性完成安装配置。喜欢使用命令行的朋友也可以在终端中输入如下命令：

```
$sudo apt-get install ubuntu-restricted-extras
```

安装成功后多数常用的插件和应用环境就搭建好了，可谓闪电配置。

(9) 配置 GCC

刚安装好的系统已经存在 GCC，但它缺少必需的头文件，所以还无法编译 C 程序，这就 需要安装 build-essential 这个软件包，安装此软件包会自动安装上 g++、libc6-dev、 linux-libc-dev、libstdc++6-4.1-dev 等一些必需的软件和头文件库。

在终端中输入如下命令：

```
$sudo apt-get install build-essential
```

当然也可以使用新立得软件包管理器安装。

(10) 安装 Qt

安装方法有 3 种：

I. 编译源码安装

请参考前面的通用安装方法，注意在 Debian 系的发行版如 Ubuntu、Kubuntu 等，需要 root 用户口令时，在你的命令前面加上 sudo 即可，而不必切换到 root 权限。如执行 make install 操作，输入：sudo make install 即可。

还有，编译安装完成后，在你机器的 home 目录下你的的用户名字的目录中, 在.profile 文件中追加如下环境变量(可以采用 vim .profile 或是 gedit .profile 的方式 打开该文件):

```
QTDIR=/usr/local/Trolltech/Qt-4.4.3/
PATH=$QTDIR/bin:$PATH
MANPATH=$QTDIR/doc/man:$MANPATH
LD_LIBRARY_PATH=$QTDIR/lib:$LD_LIBRARY_PATH
export QTDIR PATH MANPATH LD_LIBRARY_PATH
```

以上步骤执行完后，需要按 ctrl+alt+backspace 键, 注销用户,并重新登录，使更改厚的.profile 文件内容生效，其中最后两行代码不是必需的。

II. 使用图形界面安装

这种方式就是采用新立得软件包管理器来安装。

依次点击【系统】->【系统管理】->【新立得软件包管理器】，注意如果不是以 root 用户身份登录，则需要输入 root 用户密码方可使用。点击工具栏上的【Search】按钮，弹 出如图 2-59 所示对话框。



图 2-59 查找与 Qt 相关的包

在其中输入 qt 后点击【Search】按钮，系统将显示出如图 2-60 所示的结果。

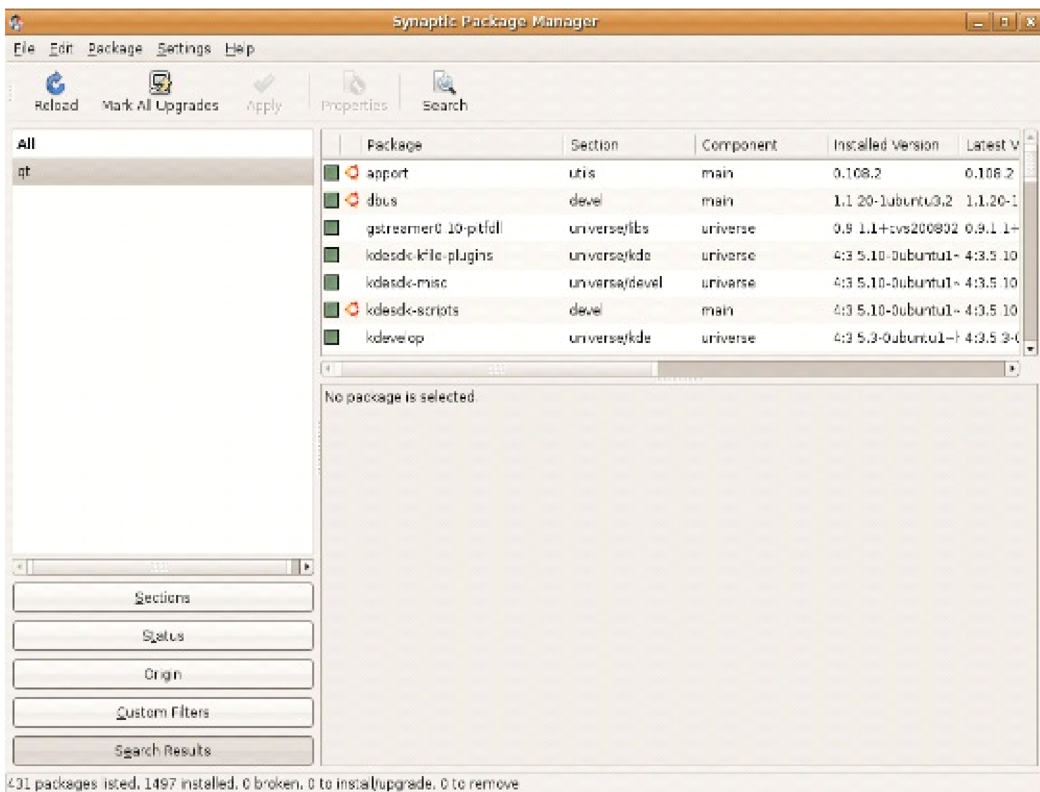


图 2-60 查找出的与 Qt 相关的结果

选中如下包：

- libqt4-core
- libqt4-debug
- libqt4-dev
- libqt4-gui
- libqt4-qt3support libqt4-sql
- qt3-qtconfig

- qt4-designer
- qt4-dev-tools qt4-doc
- qt4-config
- qt-x11-free-dbg

注意在这些软件包中，qt4-dev-tools 包含了 Qt Assistant 及 Qt Linguist 等工具； qt4-doc 是帮助文档，包含了 Qt 中各个类库的详细说明以及丰富的例子程序，可以使用 Qt Assistant 工具来打开并阅读；qt4-qtconfig 是配置 Qt 环境的工具；qt4-demos 包含很 多可以运行起来的程序范例；qt4-designer 是用来设计 GUI 界面的设计器。

点击工具栏上面的【应用】按钮，即可以安装了。安装时间视你选择包的数量和网络 速度而有不同，但通常会比采用源码编译方式的时间略短。

这之后，仍然需要配置环境变量。

III. 命令行方式

也有的朋友喜欢使用命令行，那么可以输入下面的命令：

```
sudo apt-get install qt4-demos qt4-designer qt4-dev-tools qt4-doc qt4-doc-html
```

小贴士：如果还需要其它的没有默认安装的 Qt 库，可以在命令行输入 sudo apt-get install libqt4- 然后按 tab 键自动补全，就会列出所有以 libqt4- 开头的软件包，如图 2-61 所示。

```
hongwang@hwpc:~$ sudo apt-get install libqt4-
libqt4-assistant          libqt4-opengl              libqt4-sql-psql
libqt4-core               libqt4-opengl-dev          libqt4-sql-sqlite
libqt4-dbg                libqt4-qt3support          libqt4-sql-sqlite2
libqt4-dbus               libqt4-ruby                libqt4-svg
libqt4-debug              libqt4-ruby1.8             libqt4-test
libqt4-designer           libqt4-ruby1.8-examples    libqt4-webkit
libqt4-dev                libqt4-script              libqt4-webkit-dbg
libqt4-gui                libqt4-sql                 libqt4-xml
libqt4-help               libqt4-sql-mysql           libqt4-xmlpatterns
libqt4-network            libqt4-sql-odbc            libqt4-xmlpatterns-dbg
```

图 2-61 列出所有的以 libqt4-开头的软件包

这些都可以使用一个命令搞定，而不需要自己从源码开始编译。在记不准或不知道名 字的情况下，使用 tab 键列出所有可选的软件包是一个很实用的小技巧。

小贴士：如果还有画一些数据曲线和统计图表等方面的需求，可以安装第三方的 QWT 库。同样，只需要一个命令即可完成安装：

```
sudo apt-get install libqwt5-qt4 libqwt5-qt4-dev
```

完成安装后，打开 Qt Designer，就会发现左边的 Widget 列表里面多了“QWT Widget”这一组窗口部件。

至此，在 Ubuntu 下安装 Qt4 就介绍完了，从安装操作系统到安装配置 Qt 成功，中间的过程虽然有些复杂但脉络还是清晰的，希望读者朋友能够熟练掌握。

4. OpenSUSE

(1) 版本的选择

OpenSUSE 通常每一个发行版都区分为 KDE 和 GNOME 版，但 OpenSUSE 默认采用的就是 KDE 版。从安装使用 Qt 的角度考虑，我们推荐采用 KDE 版。另外，OpenSUSE 11.1 是目前 的最新版本，无论是软件的丰富程度、系统的易用性、性能等都较 10.3 版有较大的提升， 所以我们就以 OpenSUSE 11.1 KDE 版为例，向大家讲解安装配置 Qt4 的方法。

在 OpenSUSE 11.1 上安装配置 Qt 是比较容易的。采用编译源代码和软件包管理器这两 种方式都可以，其中采用源代码编译的方式与通用方式并无二致，只需在配置 Qt 环境时注 意一下，它的配置文件的位置与其它发行版稍稍有所不同，详见《配置环境》这一节 。

(2) 安装 Qt

下面重点讲解如何从 YaST2（OpenSUSE 的软件包管理器）中安装配置 Qt4。通常需要如 下步骤：安装 OpenSUSE、更新系统、安装 Qt4、配置 Qt4 环境。

I. 安装 OpenSUSE

推荐从 LiveCD 安装，可以从 OpenSUSE 的官方网站上自由获得镜像，网址是 http://www.opensuse.org/。安装过程很简单，遵循向导的指引，几步下来就可以完成。

II. 更新系统

安装完毕后，需要确保始终在线，然后依次点击【菜单】 ->【应用程序】->【管理员 设置】，系统将启动 YaST2 控制中心，如图 2-62 所示。



图 2-62 YaST2 控制中心

在左边的大类里面选择【软件】，在右面的细分类别里面选择【在线更新】，系统将 搜索可用于更新的部件，并显示如 图 2-63 所示界面。

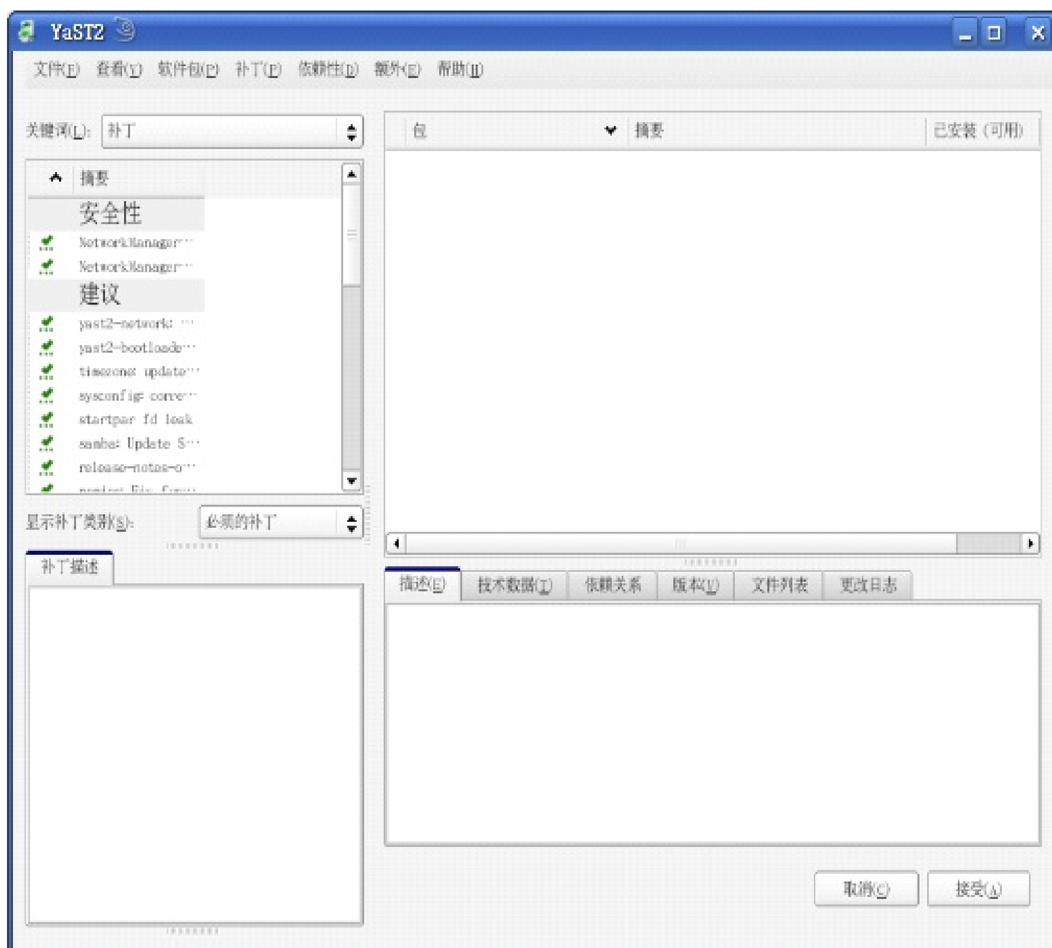


图 2-63 更新系统

在界面的左上部分选中你想要更新的构件，然后点击界面右下角的【接受】按钮即可 开始更新，需要的时间与选择的包的数量多少、机器配置情况以及网络速度有关。

III. 安装 Qt4

完成系统更新后，可以开始安装 Qt4 了。

第 1 步，启动 YaST2。依次点击【菜单】->【计算机】->【安装软件】，输入管理员密 码，然后点击【确定】按钮，如图 2-64 所示。



图 2-64 启动 YaST2

第 2 步，搜索相关软件包。在界面上的搜索栏中输入 ”qt”,搜索范围选中【名称】、【范围】选项，搜索方式选择“包含”，不选择【区分大小写】选项。然后点击【搜索】 按钮，系统将符合的选项在列表中罗列出来，如图 2-65 所示。

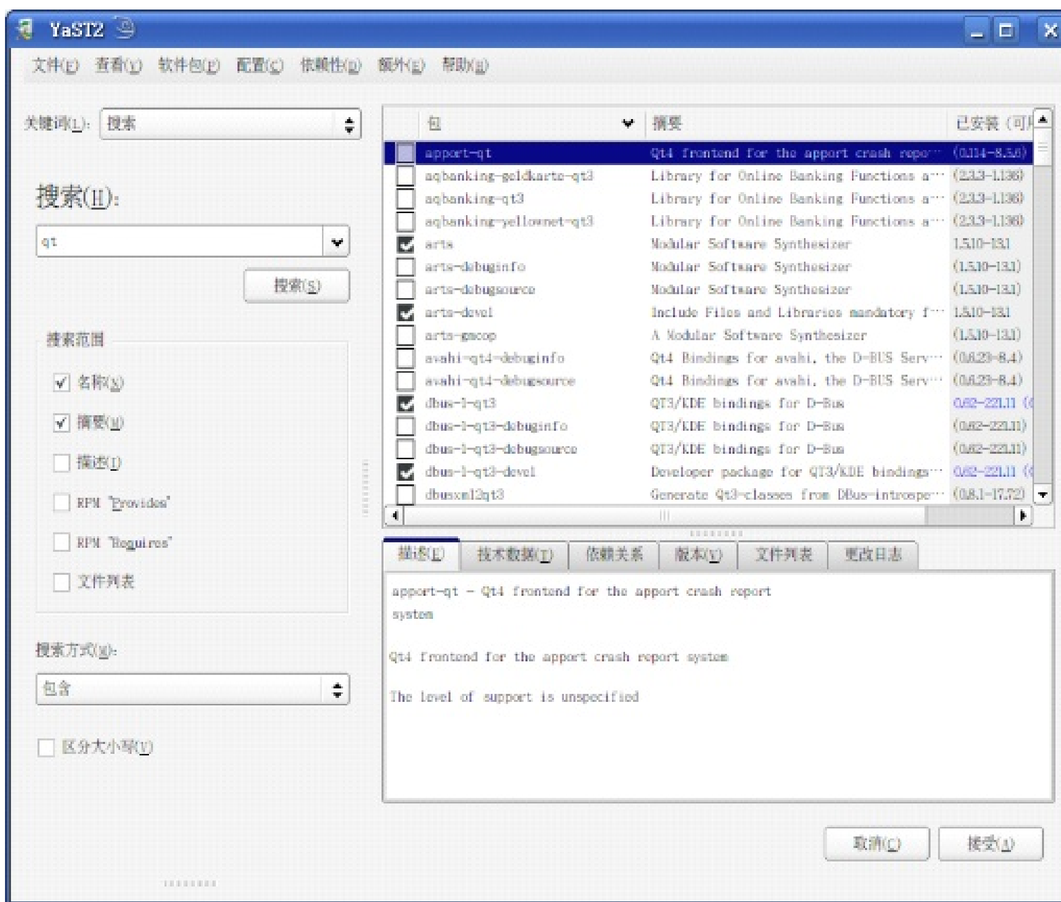


图 2-65 搜索 Qt 后的 YaST2 界面

在其中选中如下软件包：

- libqt4
- libqt4-debuginfo
- libqt4-debugsource
- libqt4-devel
- libqt4-devel-doc
- libqt4-devel-doc-data
- libqt4-qt3support
- libqt4-sql
- libqt4-sql-mysql
- libqt4-sql-postgresql
- libqt4-sql-sqlite
- libqt4-sql-unixODBC
- libqt4-sql-x11
- libQtWebKit4
- libQtWebKit4-devel
- qt4-x11-tools
- scim-bridge-qt

然后点击【接受】按钮，系统将为你安装选中的包以及所有的依赖包。通常约需要几十分钟到几小时不等的时间。

(3) 配置 Qt4 环境

这一步请参见《配置环境》一节，那里有详细的说明。 需要特别注意的是，OpenSUSE 环境变量配置文件与其它发行版有所不同 。

5. Mandriva

(1) 版本的选择

就产品的成熟度来讲，Mandriva 通常提供两个版本供使用，分别是开发版（ devel）和 正式版（official）。devel 存放的是正在开发的版本，喜欢当小白鼠的朋友常用； official 存放的是正式发行的版本。我们选择 official 版。

就授权方式来讲，Mandriva 将产品分为 FREE、ONE 和 POWERPACK 版。
POWERPACK（加强版）：这是要付费才能获得的版本，价格也不便宜，当然物有所值，它包括了技术支持、服务、以及重要的第三方私有软件，如 LinDVD、Cedega、Fluendo multimedia codecs 等，而这些东西在其它版本里是没有的。 FREE、ONE：这些都是免费的版本，其中 ONE 是 Live CD，可独立运行的光盘系统，也能够安装在硬盘上，FREE 做不到这一点。ONE 源自 POWERPACK，而且 ONE 的桌面与POWERPACK 现在一致，但是 ONE 也有较大局限：默认内核只能管理 1G 的内存。

就系统采用的桌面环境不同，Mandriva 又分为 MINI、GNOME 和 KDE 版。 MINI：迷你版本，以 ICEWM 为桌面环境（不是 icewm-light），所选软件都是轻量级软件而且数目非常少，软件包还可以定制安装，非常适合机器性能差或者极度追求速度、干净和自由的朋友。

GNOME：这是 ONE 里的一个分支，以 GNOME 为桌面环境，其实 Mandriva 默认的是 KDE， 所以不要对它抱有太高期望值，当然它也不是太差，适合喜欢 Mandriva 和 GNOME 的朋友。

KDE：这是 ONE 里的一个分支，以 KDE 为桌面环境，由于 Mandriva 默认的是 KDE，所以使用它能获得更好体验。

基于上面的描述，我们不难分析出，应该采用 Mandriva 的 ONE 的采用 KDE 的 OFFICIAL 版作为 Qt4 开发工作站的首选版本。下面就以这个版本为例介绍如何安装 Qt4.5。

(2) 安装 Qt

第 1 步，在 Mandriva 中依次点击【菜单】->【安装/删除软件】，启动软件包管理器，如图 2-66 所示。

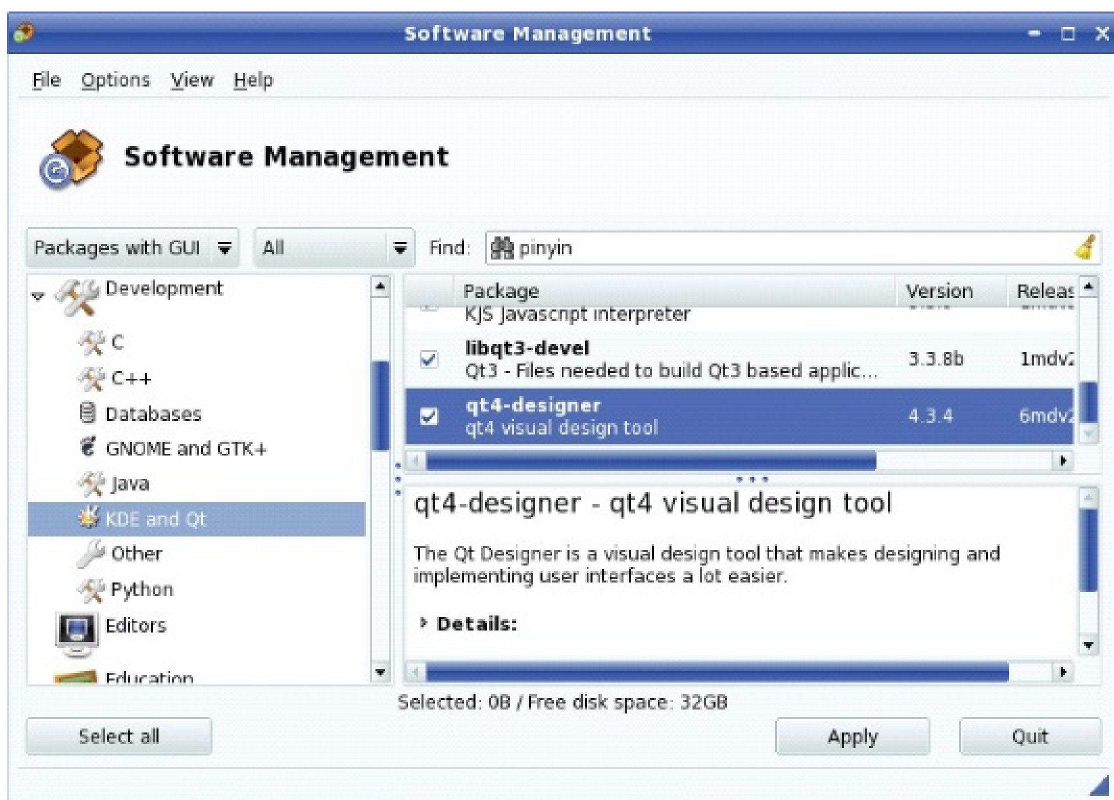


图 2-66 Mandriva 软件包管理器

第 2 步，在【搜索】框中输入"qt"关键字查询，查询条件选为【All】，系统将把所 有包含 Qt 的软件包搜索出来，如图 2-67 所示。

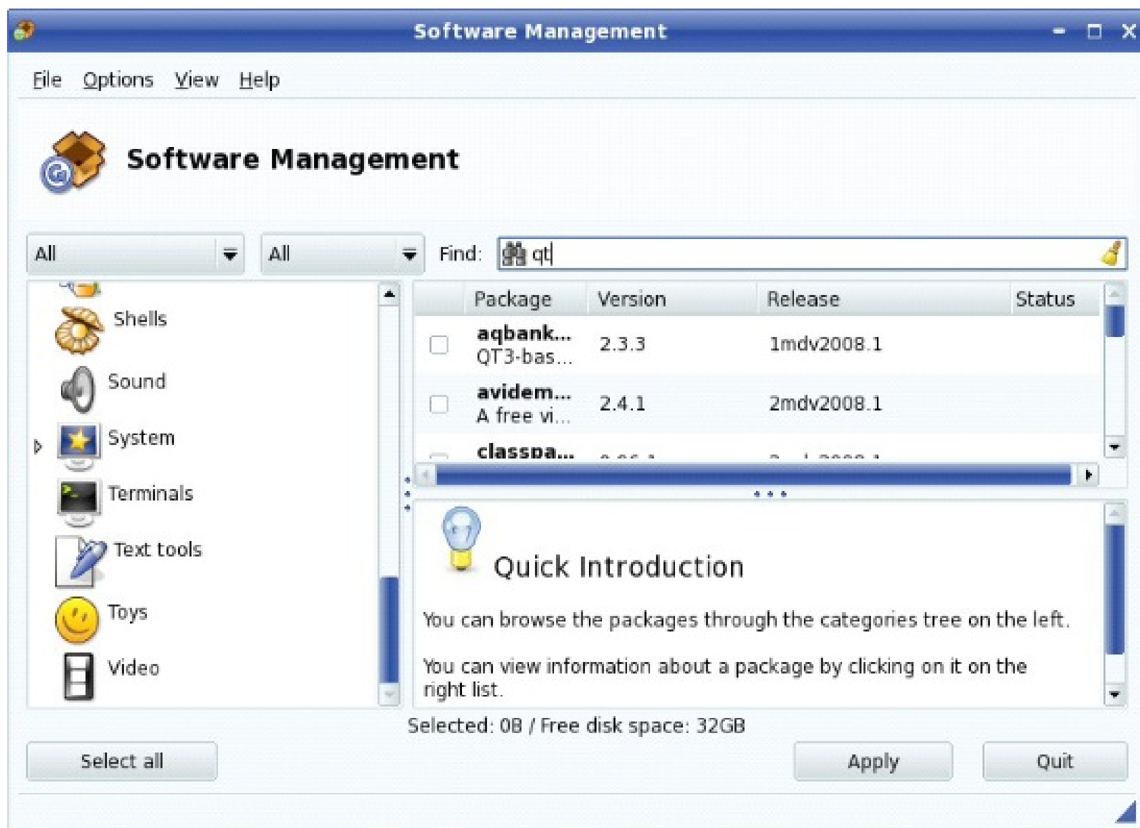


图 2-67 搜索与 qt 相关的包

选中如下软件包：

- libqt3
- libqt4-devel
- qt4-common
- qt4-designer
- qt4-doc
- qt4-assistant
- qt4-examples
- qt4-linguist
- qt4-qtconfig
- qt4-qtdbus
- qt4-qvfb
- qt4-tutorial

安装 libqt3 这个包是因为需要使用 qt3config 来配置 Qt4。

libqt4-devel 这个包含了如下包：libqt3support4、libqtcore4、libqtdbus4、libqtdesigner4、libqtgui4、libqtnetwork4、libqtopengl4、libqtscript4、libqtsql4、ilbqtsvg4、libqttest4、libqtWebKit4、ilbqtxml4、、libqt4-static-devel

此外，以下是与数据库开发相关的包，如果你不想开发数据库程序的话，也可以不安装。

- qt4-database-plugin-mysql-lib
- qt4-database-plugin-odbc-lib
- qt4-database-plugin-pgsql-lib
- qt4-database-plugin-sqlite-lib

到此为止，你的 Qt4 在 Mandirva 上已经可以使用了。 如果你有其他的需求，比如使用 Java、Python 等，还可以选装与它们相关的如 Python-qt4 包等等，这里就不再赘述了。

(3) 配置环境变量

在这里并无特殊说明，请见后面的《配置 Qt 环境》一节。

2.4 配置 Qt4 环境

笔者看到有的朋友在网上说，采用软件包管理器安装 Qt4 后，在过程中系统已经自动 设置好了 Qt4 需要的环境变量，可以不用自己手工设置了。 这是不正确的说法，经过笔者 的反复验证，在常见的 Linux 发行版中，都没有这种情况发生。也就是说 ，不论你是采用 编译源代码编译还是采用各个发行版提供的安装包的方式安装的 Qt4，你都需要配置环境变 量。下面就各个不同平台分别介绍如何配置 Qt4 环境。

2.4.1 Windows 平台

在 Windows 下打开命令行(运行 cmd 命令)，输入以下命令即可完成设置。

```
c:\&gt;set QTDIR=c:\qt-win-free-mingw-3.3.4-3
c:\&gt;set MINGW=c:\mingw
c:\&gt;set PATH=%QTDIR%\bin;%MINGW%\bin;%PATH%
c:\&gt;set QMAKESPEC=win32-g++
```

2.4.2 X11 平台

如果你在安装时，表明了 root 身份，那么你的 Qt 将被安装到 /usr/local/Trolltech/Qt-4.5.2 这个目录下面。在 X11 上，需要区分用户使用的 shell 是 哪一种。

1. 如果使用的 shell 是 bash、ksh、zsh 或者 sh 若只想某个用户用户，比如 xx（也包括 root 用户）使用 Qt，则打开 /xx/.bash_profile 文件，在其中加入以下内容：

```
# Qt4 Settings
export QTDIR=/usr/local/Trolltech/Qt-4.4.3
export PATH=$QTDIR/bin:$PATH
export LD_LIBRARY_PATH=$QTDIR/lib:$LD_LIBRARY_PATH
# End Qt4 Settings
```

注意第 1 行和最后 1 行是注释内容，用#表示。

重新登录 xx 用户，设置生效，现在 xx 用户就可以使用 Qt 了。 如果想以后新建的用户也可以使用 Qt，则在/etc/skel/.bash_profile 文件中加入与前面所述相同的内容，当然具体路径要视你的 Qt 安装路径而定。保存文件后， 别忘了重新登录 xx 用户，以使设置生效。

2. 如果使用的 shell 是 csh 或者 tcsh 需要把下面的代码添加到/etc/csh.login 文件中：

```
# Qt4 Settings
setenv PATH /usr/local/Trolltech/Qt-4.4.3/bin:$PATH
# End Qt4 Settings
```

小贴士：特别需要注意的是，这些配置文件默认情况下都是隐藏不见的，你需要设置系统来显示隐藏文件。

还有一点，就是各个发行版的配置文件可能会有不同， 现在将常见的几种发行版的配 置文件概括至表 2-6 中（以下以 xx 代指用户名）：

表 2-6 Linux 发行版的配置文件

发行版名称	采用 **bsh** 等作为 **shell**	采用 **csh** 等作为 **shell**
Fedora Core	/home/xx/.bash_profile	/etc/csh.login
OpenSUSE	/home/xx/.profile	/etc/csh.login
Mandriva	/home/xx/.bash_profile	/etc/csh.login
Ubuntu	/home/xx/.bash_profile	/etc/csh.login
Kubuntu	/home/xx/.bash_profile	/etc/csh.login
Red Hat	/home/xx/.bash_profile	/etc/csh.login
Red Flag	/home/xx/.bash_profile	/etc/csh.login
Everest	/home/xx/.bash_profile	/etc/csh.login

2.5 问题与解答

问：我在 Ubuntu8.04 上编译安装 Qt4 开源版，config 时遇到如下错误：

Qtchecking for Qt... configure: error: Qt (>= qt 3.3 and < 4.0) (headers and libraries) not found. Please check your installation!

答：这个问题是由于你还没有安装与 Qt3 相关的库文件的缘故，打开一个终端，输入

```
sudo apt-get install libqt3-mt-dev
```

安装完毕后，再重新编译安装 Qt4 就可以了。

问：商业版 Qt 在 Solaris 上编译的问题

我购买了商业版的 Qt，是 4.4.0 版本，现在升级到 4.4.1 了，在 Windows 上开 发，没有出现什么问题，现在想转到 Solaris 上面，但解压 qt-x11-commercial-src- 4.4.1.tar.gz 时总是出现问题。

首先我是用 gunzip 解压成 qt-x11-commercial-src-4.4.1.tar，然后用 tar xvf 来解压，到一半的时候就出错了，提示 "校验和错误"。我是在 windows 下下载的 qt-x11- commercial-src-4.4.1.tar.gz，这是什么原因呢？

答：原因在于 solaris 下 tar 有 bug，解压路径加文件名过长时就会出错。所以不要用 solaris 自带的 tar，而要用 gnu 的 tar, 可以在 www.sunfreeware.com 上下载到。

问：我的 Qt4 已经安装完了，但是忘记编译数据库驱动了。如果加上的话，是不是还 要重新编译一遍，很费时费力，有什么办法？

答：你还是要走一遍这个完整的编译流程，但这次它实际上不是重新编译，而是只编 译需要改变的东西，这是 make 机制决定的，你已经编译过的其他东东，这次时就会略过， 不会再次编译了。

问：编译 Qt4.5 时如何剔除一些并不想要使用的内容，比如 examples 和 demo？ 答：有几种方法可以实现，这里推荐一种，可以同时节省时间和空间，方法是在

configure 时，输入-nomake demos nomake examples，即表示不编译 examples 和 demos。

完整的命令是：

```
configure -nomake demos nomake examples
```

问：我在 linux 下都把 Qt 安装成功了，就是 make install 成功了，可是，当我编译一个 qt 的例子时候，执行 qmake，系统告诉我，没有这个命令，怎么办呀，难道安装不是 使用 make install 命令吗？

答：这种情况通常是由于你在安装 Qt 后没有设置环境变量，你把 qt 的安装目录加入 到 PATH 里面去，应该就可以了。

再次提醒，编译安装 Qt 的基本步骤如下：

1.下载

2.编译安装

3.配置环境变量

4.验证安装：qmake -v 看看输出是什么 5.减小体积，执行 make clean，将一些中间文件去掉。

其中，配置环境变量的方法是这样的，在你的 shell 配置文件里加上下面两句：

```
PATH=/usr/local/Trolltech/Qt-4.5.2/bin:$PATH
export PATH
```

2.6 总结与提高

这一章是学习 Qt4 编程的第一站，只有能够正确的安装和配置 Qt4，才可以顺利开启开 发进程。而据笔者的经验，在网上很多朋友提出的程序编译时的问题，往往与 Qt4 的安装 与配置有关。所以这是一项基本功，尤其是编译安装 Qt4 的方法必须要牢牢掌握。

第 3 章 Qt 编程基础

本章重点

- 掌握标准 C++的基础知识和技能
- 掌握不同平台上的基本编程机制和原理
- 掌握 Vi 的使用
- 掌握 GCC 和 GDB 的使用

本章主要讲述与 Qt 编程相关的一些基础知识和技能，内容涵盖了 Windows、Linux 和 Mac OS X 这三大主流平台，由于篇幅关系，有些内容无法详细展开，希望读者朋友能够查 阅相关的书籍和资料辅助学习。

对于本章内容的学习，笔者建议开始时可采用 “粗读”的方式，以对整体有一个把 握。在接下来学习本书的其他内容时，可以翻过头来对照参考，效果更佳。

3.1 标准 C++精讲

掌握标准 C++的基础知识和技能是使用 Qt 进行编程的前提，虽然 Qt 也支持其他的语言 扩展（比如 Java、Python 等），但 Qt 的基础和努力方向仍然是以 C++语言为主，所以读者 朋友一定要掌握标准 C++。

3.1.1 程序设计语言介绍

1.软件

计算机内部所有能够存储的各种数据和能够执行的各种程序都称为软件。而程序一词 经常有两种理解：（1）由程序员编写的源代码；（2）可执行的软件。

程序通常可以分为以下几类：

(1) 操作系统（Openation System）

为用户管理计算机软硬件的程序。例如 DOS、Windows 98、Windows XP、Windows CE、UNIX、Linux、Mac OS X、BSD、Solaris 等。

实际上，操作系统包含很多可执行程序，这些程序组合在一起，完成一个或几个特定 的任务。这些程序的根本目的在于有效的组织计算机的硬件资源，并为用户提供一个访问硬 件的友好界面。通常也可以将其称为系统软件。

(2) 应用软件（Applications）

在操作系统下执行的，具有特殊用处的程序，如字处理软件 Microsoft Word、游戏软件、财务软件等。这部分程序，也就是程序员主要的服务方向，也是软件设计中商业利润比较大的一部分内容。

(3) 应用程序开发环境（Application Development Environment）

协助程序员开发应用程序的特殊程序，如 Microsoft Visual Studio、Eclipse 等。

2.程序语言

程序语言是人与计算机交流的工具。通过程序语言我们可以编写程序，控制计算机执 行相应任务。到目前为止，先后出现了 4 代程序语言。

第 1 代语言：机器语言（machine language）

计算机执行的每一个操作都是由一组特定的二元指令所指定，称为操作码。通常将这些操作码称为机器语言。该语言不仅难于读写，撰写复杂，而且非常容易出错。

第 2 代语言：汇编语言（assembly language）

以接近英文和数学式的方式写作程序，它的语意与机器语言有一对一的对应关系。该语言能发挥特定机器的硬件功能，编译后的程序运行速度快、效率高，但因仍与机器硬件相关，编写仍然较困难。

第 3 代语言：高级程序语言（high -level programming language）

包括 Fortran、Pascal、C、C++、Java、Basic、C#等，用这些语言编写应用程序的时 候，通常不需要知道 CPU 执行的细节。这些语言都有自己的特色，能胜任某一方面的程序 设计。例如 Fortran 用于科学及工程的方程式运算，而 C 语言具备汇编语言的优点，可以 直接进行位运算，而且有高度的结构性，代码相对容易维护，可理解性大大提高。 C++不仅 继承了 C 的优点，而且引入了面向对象程序设计的思想。

第 4 代语言：特殊用途语言

使用在特殊的环境中，便于非专业程序设计人员使用的语言，例如 Perl、SQL、MATLAB 等，它们通常都不需要声明变量，而且有很多现成的功能可以套用。

3.编程思想

编程思想逐渐由结构化编程发展到面向对象编程。 结构化编程的主要思想是把问题细化，即把现实中的目标分解成一系列的任务，对每一个任务再进行分解，直到每个任务都能被解决为止。这是处理复杂问题的一种非常成功的方法。即使是现在，在处理很多问题上仍会经常用到这个解决问题的思想。当然，这种编程 思想存在着问题。首先，它将数据结构从函数中分离出来，这使得程序员在设计程序的时候不得不把一个事物的属性与方法分离开考虑，于是就很难真实的表现现实生活中的模型。其 次，代码的可重用性不高。虽然，现在用的很多函数都是用 C 语言编写的，但是，现在需 要的可重用的代码已经不再是这个层次上的代码，而是集中了很多功能和数据的完整的组件，并且要保证其可维护。

面向对象的编程思想满足以上的种种需求，它提供一种方法，可实现软件组件的可重 用性，并将数据与操作数据的任务结合起来。

面向对象编程的实质就是将“对象”作为模型，而不是“数据”。用作模型的对象可 以是屏幕上的界面，如按钮、列表框，也可以是现实世界中的对象，如自行车、人等。

面向对象的编程思想必须要涉及到封装、继承、多态性这 3 种技术，C++完全支持它们。

(1) 封装

实际的事物有很多组成部分，描述一个事物也可以从不同的角度出发，从程序设计的 角度出发，需要两个元素－数据和函数，当从这两个元素出发去描述一个事物时，会看到： 一个事物本身具有一些属性，相应的存在一些行为能改变其中的一些属性。要用程序中的对 象来代表这些实际的事物，于是，用数据代表事物的属性，用函数反映事物的行为。这样， 与该对象相关的数据和行为就被封装在这个对象里了。

例如：一个人有肤色、身高、体重等属性，有自己的说话、跑、跳等行为。但有时， 属性并不是实际存在的一个事物特征的反映，它可能是一种人为抽象出来的状态的反映，例 如一个人有静止、说话、跑步、睡觉 4 种状态，可以定义一个属性来表示当前这个人是处 于什么状态。假设这个人处于百米跑起跑前的静止状态，当发令枪响起，这个人开始跑（即 发出跑的动作），同时状态属性发生变化，由静止状态变为跑步状态；还有其他的属性也随 着这个动作的发出而发生变化，由静止状态变为跑步状态；还有其他的属性也随着这个跑的 动作的发出而发生变化，如心跳频率加快、移动速度变快等。这个过程即反映了行为改变属 性这一特征。

程序并不一定要完全模拟实际的事物。在系统分析时，希望设计结果能真实反映实际 问题，但是，也没有必要对涉及到的所有的事物都建立模型。下面针对系统分析过程中可能 遇到的问题举例。一个顾客染发，染发的动作会改变头发的颜色属性。显然，染发的动作由 理发师发出，但在程序中是否真的由理发师发出这个动作还需要根据具体问题再进行分析。

I. 当具体问题中理发师的存在仅仅完成一个染发的作用，那么要考虑理发师是否有存 在的必要，如果没必要，就果断的删除这个对象，把染发的动作交给顾客自己来完成。

II. 当具体问题是为了描述理发师的行为状态变化，那么可能就把这个染发的动作交给 理发师来完成。

该例子也说明，行为交给哪个对象来完成，是需要就具体问题进行分析而得到的，关 于这个问题可以参看下列规则：

- 如果行为影响或修改了一个对象，最好把该行为交给目标对象（而不是动作的发起 者）。
- 如果行为涉及到多个对象以及它们之间的关联，从中找出处于中心地位的对象，将 行为将给该对象完成。

这两个规则只是经验的反映，经验不能反映所有的情况，也有可能把行为交给其他对象来完成也是合理的，这需要根据实际情况进行判断。

(2) 继承

这是代码重用的很有效的方法，新的类可以通过继承原有类，并选择性的增加或修改其中的属性或行为，以达到利用原有类的目的。现介绍一下代码重用的方式。

注意，类是与对象相关联的概念，将在稍后介绍。

I. 源代码剪贴

最原始的形式，缺点很多。首先是复制或修改原有代码可能会出错，其次需要对源代码有一定程度的了解。另外，存在严重的配置管理问题，人们几乎无法跟踪源代码多次修改重用的过程。

II. 源代码包含

许多程序语言都提供了包含库中源代码的机制。使用这种重用形式时，配置管理问题有所缓解，因为修改了库中源代码之后，所有包含它的程序自然都必须重新编译。

III. 继承

利用继承机制重用类库中的类时，无需修改已有的代码，就可以扩充或具体化原有类，因此，基本上不存在配置管理的问题。

(3) 多态

允许使用相同的接口，与各种不同的派生类定义出来的对象交互，能够产生正确的行为。

例如，中国人说汉语，美国人说英语。“说”的动作相同，却有着不同的内容－汉语和英语；对于不同类型的电视机，都可以使用相同的动作：单击【播放】按钮，开始播放电 视频道，显然不同类型的电视机显示的方法会有所区别，但没有必要关心这些，人们看电视的目的已经达到了。

3.1.2 C++语法基础知识

1.预处理知识

在阅读本节之前，读者最好看看 C++这方面的专著。这里只讲一些常用的知识，并不对 C++作全面深入的讲解。

C++程序由对象、函数、变量及其他组件组成。 从最简单的程序讲起：

```
#include &lt;instream.h&gt;
int main()
{
    cout&lt;&lt;”Hello World\n”;
    return 0;
}
```

这是一个 Console 程序，撰写 Console 程序时需要注意：主程序为 main 可以使用 C Runtime 函数和不涉及 GUI 的 Win32 API 函数。撰写 Console 程序是学习 C++的第一步。进入 Console 模式进行编程，使用 VC++ 6.0，操作步骤如下：

(1) 选择【File】菜单中的【New】菜单项，在弹出的对话框中选择【Projects】标 签；再选择“Win32 Console Application“程序，然后输入工程名称， 单击【OK】按钮； 在接下来的对话框中， 为了脱离 VC 提供的代码支持，选择“an empty project”，然后单 击【Finish】按钮，会得到一个不包含任何工程文件的工程。之后，要加入包含着主函数的 头文件。

(2) 选择【File】菜单中的【New...】子菜单，在弹出的对话框中选择【Files】标 签，再选择“C++ Source File”，选中“Add to Project”项，选中（1）中的工程名 称。接下来，确定文件的名称，单击【 OK】按钮，于是就得到了需要的文件，同时可以在这 个空文件里键入上述代码。

(3) 运行程序，它会弹出一个对话框提示没有可执行程序，询问是否创建可执行程序。 单击【是】即可，然后 VC 开始编译，链接目标文件。如果没有意外的话，会得到一个 DOS 窗口，窗口内有如下的输出：

```
Hello World
Press any key to Continue
```

最后一句话并不是源代码反映的内容，这是由编译器提供的，方便查看输出内容的提

示。如果读者想看程序真正的输出结果，就进入命令行下（即以前的 DOS 窗口下），运行该程序，读者就可以看到 Hello World，而没有其他的信息。

```
#include&lt;iostream.h&gt;
```

该语句的意思是：将文件包含到当前的文件中，符号 #是预处理 标志。

注意，每次启动编译器时，先运行预处理器。预处理器浏览源代码，找到以 #开头的 行，处理这些行，预处理的作用是改变源代码的文本。结果生成一个新的源代码文件——个 通常看不到的临时文件，但读者可以指定编译器保存它，这样，读者可以在感兴趣的时候或 者需要的时候检查它。编译器不读取最初的源代码文件，它读取预处理器输出的结果并编译 该文件，最终生成可执行程序。

初步介绍预处理作用如下，以便读者了解预处理的概念：

I. 包含另一个文件（通常是头文件），为了引入需要的代码。

II. 定义符号，祈祷开关作用，可以根据机器的情况、操作系统的情况及用户的需求来 决定哪部分代码有效，例如，如果要在 Win32 的环境下编程，那么，就定义 Win32 这个符 号。

III. 定义宏，简化常用的数据，或者简化复杂的函数声明、定义的过程。

预处理的功能不止这些，以上是本书程序中略有涉及的内容。

正式的程序从 main 函数开始，每一个 C++程序都有一个 main 函数。函数是指能实现一 个或多个功能的代码块。通常函数是由其他函数调用或激活，而 main 属于特殊情况。程序 在开始的时候自动调用 main。从本质上讲，也是被调用，不过，那些都不是现在需要关心 的事情。

所有的函数都以左大括号开始，以右大括号结束。两个大括号之间是函数体。 使用对象 cout 将一个字符串打印到屏幕上。最后返回 0。

这些就是一个 C++程序的基本轮廓。下面开始讲解+的基本语法。 2.基本数据类型

在任何一台计算机中，每种变量类型都占据一定量的内存单元。但并不是每种变量类 型占的内存大小在每台计算机上都相同，一个整型变量在一台机器上可能是 2 个字节，而在另一台机器上可能是 4 字节，但对任意一台机器而言，这个值是确定的。字符型变量通常只有 1 个

字节。

在大多数计算机上，短整型是 2 个字节，长整型是 4 个字节，而整型可能是 2 个字节也可能是 4 个字节。编程语言并没有对此做出精确的定义，它定义的是短整型必须小于或 等于整型的大小，整型必须小于或等于长整型的大小。

整型的大小是由读者所使用的处理器（ 16 位还是 32 位）和编译器决定的。例如，在使 用 Visual C＋＋4.0 及以上版本的 32 位 Intel x86 计算机上，整型为 4 字节。详细的变量 类型说明如表 3-1 所示。

表 3-1 C++变量类型说明

类型	大小	值
bool	1 字节	true 或 false
unsigned short int	2 字节	0~65535
short int	2 字节	-32768～32767
unsigned long int	4 字节	0~4294967295
long int	4 字节	-2147483648~2147483647
int（16 位）	2 字节	－32768～32767
int（32 位）	4 字节	-2147483648~2147483647
unsigned int（16 位）	2 字节	0～65535
unsigned int（32 位）	4 字节	0~4294967295
char	1 字节	256 个字符
float	1 字节	1.2e-38~3.4e38
double	8 字节	2.2e-308~1.8e308

3.表达式和语句

在 C++中，语句用于控制程序的执行顺序、计算表达式的值，或者什么都不做（空语 句）。

所有的 C++语句都以分号结尾，即使是空语句也是如此。 下面是典型的语句：

```
value = value2 – value4;
```

在 C++中任何一个计算值的操作都可称为表达式，表达式总是能够返回一个值。表达式可以简化到 1，2，3 这样的整数，也可以是 a+b 这样的形式，重要的是它能返回一个值。

运算符是一种能使编译器进行某项操作的符号。运算符作用于操作数，在 C++中所有操 作数都是表达式。

(1) 赋值运算符（=）：使赋值运算符左边的操作数的值改变为赋值运算符右边的值。 例如：

```
value = 20;
```

(2) 数学运算符：加（+）、减（-）、乘（*）、除（/）、取模（%）。取模运算符（%）就是求整型除法的余数。

I. 一些特殊的运算符：如，`+=`，`-=`，`*=`，`/=`，`%=`。 例如：

`a+=2;` 等价于 `a=a+2;`

`a-=2;` 等价于 `a=a-2;`

`a*=2;` 等价于 `a=a*2;`

`a/=2;` 等价于 `a=a/2;`

`a%=2;` 等价于 `a=a%2;`

II. 单目运算符：如++、--等。

`a++;` 和 `++a;` 等价于 `a=a+1;`

`a--;` 和 `--a;` 等价于 `a=a-1;`

`x=a++;` 等价于两步操作：先是 `x=a;` 然后是 `a=a+1;`

`x=++a;` 等价于两步操作：先是 `a=a+1;` 然后是 `x=a;`

在复杂的表达式之中，多个运算符同时出现，不同的符号有不同的执行优先级。例如：a=8-32，对于该表达式，由于号的优先级大于-号，所以先执行 32，然后执行 *8-6,*最后执行 *a*=2。可以通过添加括号（）来改变运算顺序。将原式改成 *a=(8-3)*2，这时，先 执行 8-3，再执行 5*2，最后执行 a=10。

(3) 关系运算符：用来对两个量进行比较的（等于、大于或小于）符号。每条关系语句 的值要么为真要么为假，每个表达式都可以按真假来求值。凡是数值运算结果为 0 的表达 式返回假，否则返回真，也就是返回 False 或 True。C++基本的关系运算符如表 3-2 所示。

表 3-2 关系运算符说明

名称	运算符	例子	值
等于	==	1==1	true
		1==2	false
不等于	!=	1!=1	false
		1!=2	true
大于	>	1>2	false
		2>1	true
大于等于	>=	1>=2	fasle
		1>=1	true
小于	<	1<2	true
		2<1	false
小于等于	<=	1<=2	true
		2<=1	false

(4) 逻辑运算符：用于判断操作数之间逻辑关系的运算符， C++基本的逻辑运算符如表3-3 所示。

表 3-3 逻辑运算符

运算符	符号	例子
与（AND）	&&	表达式 1&&表达式 2
或（OR）	||	表达式 1||表达式 2
非（NOT）	!	!表达式

(5) 条件运算符（?:）

条件运算符（?:）是 C++中唯一的三目运算符。其格式如下：

```
（表达式 1）?（表达式 2）:（表达式 3）
```

它的意思是：如果表达式 1 的值为真，就返回表达式 2 的值；否则就返回表达式 3 的值。通常，这个值将赋给某个变量。

例如：

```
c=(a&gt;b)?a:bj;
```

等价于

```
if(a&gt;b)
{
    c = a;
}
else
{
    c=b;
}
```

4.函数

函数实际上是能够对数据进行处理并返回一个值的子程序。每个 C++程序都至少有一个 函数 main。当程序启动时，系统自动调用 main 函数。main 函数可调用其他的函数，其中 一些函数还可以再调用其他函数。每个函数都有自己的名字，当程序读到函数名时，程序执 行就转到函数体。这个过程称作“调用函数”。当函数执行完后，程序又跳回到函数名所 在行的下一行继续执行。设计得好的函数能执行特定的易于了解的任务。对于复杂的任务， 应该将其分成多个函数来完成，这些函数可以被程序依次调用。

函数通常有两种类型：用户定义函数和内置函数。用户定义函数是由用户自己编写的 函数。内置函数则是编译器软件包的一部分－由开发商提供给用户使用。

(1) 函数的声明

在使用函数时，必须先声明再定义。声明告诉编译器该函数的名称、返回值类型以及参数。定义则告诉编译器该函数的功能是什么。如果不声明，任何函数都不能被其他函数调用。函数的声明又称为函数原型。

有 3 种函数的声明：

I. 将函数原型写在某个文件中，再用#include 将其包含到程序中

II. 将函数原型写到使用该函数的文件中

III. 在函数被其他函数使用前定义该函数，这样做时，函数定义将作为声明。

实际上，大家使用的许多内置函数已经将它们的函数原型写到了用 #include 包含在程序使用的头文件内。对于读者自己编写的函数，必须包含该原型。函数原型也是一条语句，也就是说它以分号结尾。它由函数的返回值类型和函数标识组成。函数标识包括函数名和参 数列表。参数列表包含函数的所有参数及其类型的列表，这些参数由逗号分开。

函数原型与它的定义必须在返回类型和标识上完全相符。如果不相符，就会有编译错 误。不过，请注意，函数原型中不必包含参数名，而只需要参数类型。例如，像这样的函数 类型就完全合乎要求：

```
long Area(int,int);
```

这个原型声明了一个函数 Area，它返回一个长整型变量，它有两个参数均为整型。不过，虽然这种方式合法，但它不够好。如果在原型中加入参数名，这个函数原型就会变得清 晰得多。例如：

```
long Area(int length,int width);
```

就可以比较清楚的看到函数的作用，参数的意义。所有的函数都有返回值，如果未明确声明返回值类型，则系统自动默认为整型。不过，精确的声明每个函数的返回值类型，包 括 main，可使程序更清楚。

(2) 函数的定义

函数的定义由函数头和函数体组成。函数头与函数原型很像，只是其中的参数必须有名称，而且函数头不以分号结尾。

函数体是包含在一对大括号内的一组语句。 可以这样使用函数：

```
int main()
{
    int areaOfRect = Area(4,5);
    cout&lt;&lt;areaOfRect;
}
```

函数可以调用其他函数，甚至调用自己。

5.类和对象

类与对象的关系反映在现实生活中就像是概念与实体、抽象与具体的关系。例如：提 起自行车，一定能在脑海中形成一个概念－两个轮子，可以用脚蹬的交通工具。当然，大家 不可能“骑着”这个脑海中的概念去工作，或上学，只能骑着一部真正客观存在的自行车 去工作或上学。这就是类与对象的区别，就是说对象是类的个体实例。

程序通常是用来解决实际问题的，为了方便模拟现实存在的事物以及事物之间的关 系，需要建立这样的类来模拟现实生活中相应的事物。例如：模拟猫捉老鼠的 过程，就需 要建立两个类，一个模拟猫，一个模拟老鼠。猫有个动作是追赶，老鼠有个动作是逃跑。这 样就可以动手模拟这个过程了。

要声明一个类，需要使用 C++的关键字 class，然后列出类的成员变量和成员函数，注 意类的声明要以分号结束，例如：

```
class Rect
{
    int width;
    int length;
    void show();
    int getArea();
};
```

声明 Rect 时，并没有为它分配内存。它只是告诉编译器：Rect 是什么，它包含什么数据（width,length），能做什么（show(),getArea()）。他还告诉编译器 Rect 要占多大空 间，大家可以算一算，Rect 需要编译器预留多大空间呢？答案是 8 个字节，width 和 length 各占 4 个字节，加起来是 8 个，系统不会为成员函数分配存储空间。

在定义类的成员变量和成员函数时，还经常用到两个关键字 public 和 private。 类的所有成员默认时均为私有。私有成员只能通过类本身提供的成员函数内访问。公有成员则可以被该类的对象访问。如上例的 Rect 类，在默认情况下，width 和 length 都是私有成员变量，所以如果你写下了这样的代码，就会出错：

```
Rect rect;
rect.width = 10;
rect.length = 10;
```

因为在默认情况下成员变量 width 和 length 都是私有的，如果想让这两个成员变量都能够被访问，可以将它们改变为共有的，声明如下：

```
class Rect
{
public:
    int width;
    int length;
    void show();
    int getArea();
};
```

在 public 之后修饰的成员变量和函数都被定义为公有的。这样，下列语句就是合法

的：

```
rect.show();
```

要定义类的成员函数，可以在类声明函数的地方，也可以在类声明体之外的地方，但

需要 include 包含类的声明的文件。

```
#include”Rect.h”
int Rect::show()
{
    cout&lt;&lt;width;
    cout&lt;&lt;length;
}
```

类中的成员函数类似 C 函数的定义方式，不同的是，每个成员函数之前带有“类名”和::域作用符，以表示该成员函数是属于哪个类的成员。

当然，私有的数据成员和函数并不是只有该类的成员函数才能访问到，友元函数和友 元类也能访问，例如：

class Rect { public: int width; int length; void show(); int getArea(); friend void setRectWidth(Rect* pRect); friend class Point; };

这样类 Point 和函数 setRectWidth 都可以对类 Rect 的私有成员变量和函数访问了。

6.类的继承机制

C++中允许单继承和多继承。一个类可以根据需要生成派生类。派生类根据情况继承了基类的方法，还可以定义新的方法。一个子类的每个对象包含有从父类那里继承来的数据成

员以及自己所特有的数据成员。在 C++语言中，派生类可以从一个基类派生，称为单继承；也可以从多个基类派生，就是所谓多继承。

派生类的继承方式有“公有继承（public）”、“私有继承（private）”和“保护继 承（protected）”这 3 种常见的方式，此外还有虚继承这种方式。

这里重点介绍一下公有继承。 公有继承的特点是基类的公有成员和保护成员作为派生类的成员时，它们都保持原有的状态，而基类的私有成员仍然是私有的。

(1) 基类成员对其对象的可见性：公有成员可见，其他不可见。这里保护成员同于私有 成员；

(2) 基类成员对派生类的可见性：公有成员和保护成员可见，而私有成员不可见。这里 保护成员同于公有成员；

(3) 基类成员对派生类对象的可见性：公有成员可见，其他成员不可见。

所以，在公有继承时，派生类的对象可以访问基类中的公有成员；派生类的成员函数可以访问基类中的公有成员和保护成员。例如：

```
class BaseClass
{
protected:
    int x;
    int y;
};

class DerivedClass:public BaseClass
{
    void show(){cout&lt;&lt;x;cout&lt;&lt;y;}; //访问基类中受保护的数据成员
};
```

本节讲述了 C++程序设计的基本内容，简单的讲解了 C++中实现面向对象程序设计思想

的语法、类与对象，这些只是 C++程序中很少的一部分，希望读者能够阅读相关书籍加以巩固提高。

3.1.3 C++高级应用－虚函数

这里重点讲一下虚函数。虚函数就是人们希望在派生类中被重新定义的函数，当我们

用基类的指针指向派生类的对象时，就能调用该派生类的虚函数。例如：

```
class BaseClass
{
public:
    virtual void show(){};
};

class DerivedClass:public BaseClass
{
public:
    int x;
    int y;
    void show(){cout&lt;&lt;x;cout&lt;&lt;y;};
};
```

函数 show()被基类声明为虚函数，那么在派生类中也就都是虚函数。使用虚函数的用

意是什么呢，请看：

```
BaseClass* pBaseObject = NULL;
DerivedClass DerivedObject;
pBaseObject = &DerivedObject;
pBaseObject-&gt;show();
```

运行后，屏幕上将显示 x,y 的值。

该例中，先定义了一个基类的指针，又定义了一个派生类的 对象，接着，用基类的指 针指向派生类的对象，最后用基类的指针调用函数 show()，这时，结果将输出 x,y 的值。

从结果上可以看出，基类的指针调用了派生类的成员函数。

需要注意以下几点： 1.在基类中声明一个成员函数为虚函数后，在它的派生类中此成员函数也是虚函数，

并且不需要在前面加关键字。

2.当指针调用函数时，如果调用的是虚函数，则根据指针指向的对象访问函数；如果 调用的是非虚函数，则指针的类型调用相应的函数；如果虚函数在派生类中没有定义，则会 自动调用基类中的函数定义。

另外，虚函数的使用需要谨慎，因为它会增加一些额外的开销，不过这点开销不足以 削弱它的强大功能，除非用户漫无目的滥用它。

此外，可以将一个虚函数声明为一个纯虚函数：

```
virtual void show()=0;
```

这么做，等于告诉编译器在对象中为函数 show 保留一个间隔，为将来从基类中派生的

函数占据一个位置。纯虚函数有以下特点：

1.纯虚函数仅仅是用来为基类的派生类中的函数保留位置

2.纯虚函数在基类中没有定义，它们被初始化为 0

3.当虚函数变成纯虚函数时，任何派生类都必须给出它自己的定义。否则编译出错。 在使用中，不能创建一个带有纯虚函数的类的对象，但允许声明含有纯虚函数的类的指针。 在程序中往往会用到这个功能。

3.2 Windows 编程基础

3.2.1 你需要掌握的技能

下面是笔者总结的在 Windows 下做开发需要掌握的一些基本技能，这些都是在实际工 作中经常会用到的，在后面的章节中，我们有重点的讲解相关内容，但这并不是全部 。

- 了解 Windows 系统各个版本的特点，能够根据需求选用合适的版本；
- 熟悉 Windows 的运行机理和编程模式
- 能够熟练配置环境变量，了解注册表的功用，能够修改注册表项
- 能够熟练配置网络连接，包括局域网和 Internet
- 掌握常见的 DOS 命令，熟悉命令行的使用
- 能够熟练使用 MSDN，获取帮助
- 能够熟练使用一种代码编辑器，如记事本、 emacs 等，推荐掌握 Mingw 和 Visual Studio 的使用
- 了解 Windows 系统下打包软件的常用方法，掌握至少一种打包软件的使用

3.2.2 Windows 运行机理

要想熟练掌握 Windows 应用程序的开发，我们首先需要理解 Windows 平台下程序运行 的内部机制。

1.API 与 SDK

我们在编写标准 C 程序的时候，经常会调用各种库函数来辅助完成某些功能：初学者 使用得最多的库函数就是 printf 了，这些库函数是由你所使用的编译器厂商所提供的。在 Windows 平台下，也有类似的函数可供调用。不同的是，这些函数是由 Windows 操作系统本 身提供的。

Windows 操作系统提供了各种各样的函数，以方便我们开发 Windows 应用程序。这些函 数是 Windows 操作系统提供给应用程序编程的接口（ Application Programming Interface），简称为 API 函数。我们在编写 Windows 程序时所说的 API 函数，就是指系统 提供的函数，所有主要的 Windows 函数都在 Windows.h 头文件中进行了说明。

我们经常听到 Win32 SDK 开发、Qt SDK 开发等等说法。那么什么是 SDK 呢？SDK 的全 称是 Software Development Kit，中文译为软件开开发包。比如我们现在要开发与短信猫 相关的通信程序，在购买短信猫的同时，厂商会提供短信猫的 SDK 开发包，以方便我们对 短信猫的编程操作。这个开发包通常都会包括短信猫的 API 函数库、帮助文档、使用手 册、辅助工具等资源。也就是说， SDK 实际上就是开发所需资源的一个集合。

需要明确的是，API 和 SDK 是一种广泛使用的专业术语，并没有专指某一种特定的 API 和 SDK。

Windows 操作系统提供了 1000 多种 API 函数，作为开发人员，要全部记住这些函数调 用的语法几乎是不可能的。那么我们如何才能更好的去使用和掌握这些函数呢？答案是熟练 使用 MSDN，我们将在后面详细为大家介绍。

2.窗口和句柄

(1) 窗口

窗口是 Windows 应用程序中一个非常重要的元素，一个 Windows 应用程序至少要有一 个窗口，称为主窗口。窗口是指屏幕上的一块矩形区域，是 Windows 应用程序与用户进行 交互的接口。利用窗口，可以接收用户的输入以及显示输出。

一个应用程序窗口通常都包含标题栏、菜单栏、系统菜单、最小化框、最大化框、可 调边框，有的还有滚动条。一个典型的窗口如图所示。

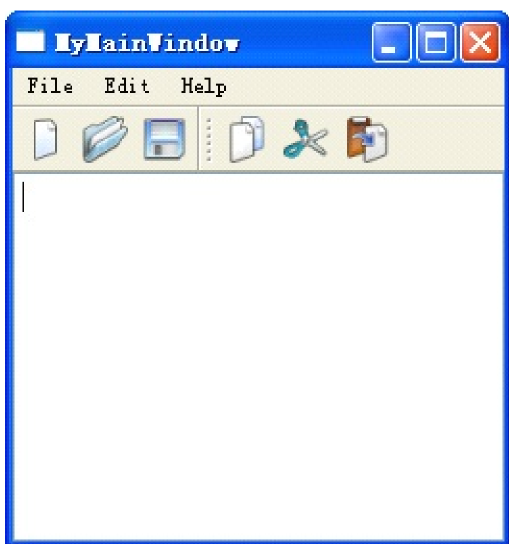


图 3-1 一个典型的窗口

窗口可以分为客户区和非客户区，客户区是窗口的一部分，应用程序通常在客户区中 显示文字或者绘制图形。标题栏、菜单栏、系统菜单、最小化框和最大化框、可调边框统称 为窗口的非客户区，它们由 Windows 系统来管理，而应用程序则主要管理客户区的外观及 操作。

窗口可以有一个父窗口，有父窗口的窗口称为子窗口。除了图所示类型的窗口之外， 对话框和消息框也是一种窗口。在对话框上通常还包含许多子窗口，这些子窗口的形式有按 钮、单选按钮、复选框、组框、文本编辑框等。

此外，我们在启动 Windows 系统后，看到的桌面也是一个窗口，称为桌面窗口，是位 于最上层的窗口，由 Windows 系统创建和管理。

(2) 句柄

下面再说说句柄（handle）。Windows 具有很强的面向对象特性。Windows 对象有很 多，譬如桌面、读取所使用的程序等等。那么，如何区分这些东西呢？答案是使用句柄。句 柄是引用不同 Windows 对象的方式。可以使用 Windows 的句柄、文件的句柄、分配内存的 句柄、

图像的句柄等等。系统在创建这些资源时会为它们分配内存，并返回标识这些资源的 标识号，这就是句柄。实际上我们也可以将这些句柄看作指针。

在使用句柄之前，必须先创建它们，当不再使用时，应当及时销毁它们。如果不销毁 它们，最终将导致资源泄露（resource leak），资源泄露有可能导致系统崩溃，所以，务 必确保在适当的时候销毁不再使用的句柄。

在 windows 应用程序中，窗口是通过窗口句柄（ HWND）来标识的。我们要对某个窗口 进行操作，首先就要得到这个窗口的句柄，这就是窗口和句柄的联系。

3.消息与消息队列

Windows 程序设计是一种基于消息的事件驱动方式的设计模式，完全不同于传统的 DOS

方式的程序设计方法。在 Windows 中，编程的骨架都是响应和发送消息。例如，当用户在

窗口中画图的时候，按下鼠标左键，此时操作系统会感知这一事件，于是将这个事件包装成一个消息，投递到应用程序的消息队列中，然后应用程序从消息队列中取出消息并响应。在

这个处理过程中，操作系统也会给应用程序 “发送消息”。所谓“发送消息”，实际上是 操作系统调用程序中一个专门处理消息的函数，称为 “窗口过程”。

(1) 消息

在 windows 程序中，消息是由 MSG 结构体来表示的。MSG 结构体的定义如下：

```
typedef struct tagMSG
{
    HWND hwnd;
    UINT message;
    WPARAM wParam;
    LPARAM lParam;
    DWORD time;
    POINT pt;
}MSG;
```

该结构体中各成员变量的含义如下：

hwnd 表示消息所属的窗口。我们开发的程序都是窗口应用程序，消息一般都是与某个 窗口相关联的。在 Windows 程序中，用 HWND 类型的变量来标识窗口。

message 变量指定了消息的标识符。在 Windows 中，消息是由一个数值来表示的，不同 的消息对应不同的数值。但是由于数值不便于记忆，所以 Windows 将消息对应的数值定义 为 WM_XXX 宏（WM 是 Window Message 的缩写）的形式，XXX 对应某种消息的英文拼写的大 写形式。例如，鼠标左键按下的消息是 WM_LBUTTONDOWN，键盘按下消息是 WM_KEYDOWN，字 符消息是 WM_CHAR 等等。在程序中，我们通常都是用 WM_XXX 宏的形式来使用消息的。

此外，我们可以定义自己的消息，并给窗口发送这些消息，您完全不用担心如何使这 些消息与代码联系起来，因为这是应用程序框架的事情。但是另一方面，这也在一定程度上 固定了程序设计上的一些结构。

wParam 和 lParam 用于指定消息的其他附加信息。比如，当我们收到一个字符消息的时候，message 成员变量的值就是 WM_CHAR，但用户输入的是那些字符，就由 wParam 和 lParam 来说明。wParam、lParam 表示的信息随消息的不同而有变化。

time 和 pt 分别表示消息投递到消息队列的时间和鼠标的当前位置。

(2) 消息队列

每一个 Windows 应用程序开始执行后，系统都会为该程序创建一个消息队列，这个消 息队列用来存放该程序创建的窗口的消息。 Windows 将产生的消息依次放入消息队列中，而 应用程序则通过消息循环不断从队列中取出消息，进行响应。这种消息机制，就是 Windows

程序运行的基本机制。

4.窗口句柄和消息 现在我们将消息与句柄联系起来。假如有一个窗口，且拥有该窗口的一个句柄（称作一个 HWND），我们命名该句柄为 your_HWND。假设因为其他的窗口刚刚从该窗口上移走，那么操作系统希望重绘这个窗口。Windows 将传递如下所示消息：

```
PostMessage(your_HWND,WM_PAINT,0,0);
```

这个函数通过句柄 your_HWND 给窗口发送了一条绘制消息。最后两个参数用作消息的额外信息，暂时可以不必深究它们的具体细节。

现在，应用程序中有一个函数用一个庞大的 case 语句来处理所有信息。例如：

```
void HandleTheMessage()
{
    switch(Message)
    {
        case WM_PAINT:
            DrawWindow();
            break;
        case WM_KEYDOWN:
            break;
        ......
    }
}
```

以上就是 Windows 中的消息和句柄的大致工作过程。了解这些后原理后，下面就可以

学习一下有关主程序以及窗口创建的知识。

5.主程序－WinMain 函数

在 windows 操作系统下，用 C 或者 C++来编写 MS-DOS 应用程序时，最起码要有一个 main 函数。当用户运行该应用程序时，操作系统会自动调用 main。但当编写 Windows 应用 程序时，就一定要有 WinMain 函数，因为当用户运行该程序时，操作系统首先调用程序中 的 WinMain 函数。该函数一般用来完成某些特殊的任务，其中最重要的任务就是要创建该 应用程序的“主窗口”。许多 Windows 集成开发环境，包括使用 Microsoft MFC 类库的 Visual C++，都通过隐藏 WinMain 函数及构造消息－控制机制来简化编程。虽然使用 MFC 编程不再需要过多关注 WinMain 函数，但是弄清楚操作系统与程序之间的这种关系是最基 本的要求。

注意，有关于 Win32 API 编程基础的内容，大家可以参看候俊杰著的《深入浅出 MFC（第二版）》一书。

6.创建窗口

Windows 窗口在创建之前，其属性必须设定好，所谓属性包括类的名字、图标、光标及窗口过程处理函数等属性。为了设定这些属性， Windows 要求注册窗口类，一经注册，就可 以创建更多的同类窗口，无需再次注册。窗口类仅仅定义了窗口的特征，所有创建窗口的对象都用窗口类来创建窗口。程序必须在产生窗口前先利用 API 函数 RegisterClass 设定属 性，这一个过程就是注册窗口类。

窗口注册完之后，就可以创建相应的窗口。 注册窗口时，必须给函数传递一个指针，这个指针指向一个包含窗口属性的结构。该结构有 2 个版本，WNDCLASS 和 WNDCLASSEX，前者本来用于 Windows 早期版本，但现在仍可沿用；后者用于 32 位 Windows，该结构包含 1 个 cbSize 成员和 1 个指向小图标的句柄，其 它两者相同。

WNDCLASSEX 定义如下：

```
typedef struct_WNDCLASSEX
{
    UINT cbSize;
    UINT style;
    WNDPROC lpfn WndProc;
    int cbClsExtra;
    int cbWndExtra;
    HANDLE hInstance;
    HICON hIcon;
    HCURSOR hCursor;
    HBRUSH hbrBackground;
    LPCTSTR lpszMenuName;
    LPCTSTR lpszClassName;
    HICON hIconSm;
}WNDCLASSEX;

ATOM RegisterClassEx
(
    CONST WNDCLASSEX *lpwcx
);
```

调用过程如下：

```
WNDCLASSEX wcex;
wcex.cbSize = sizeof(WNDCLASSEX);
wcex.style = CS_HREDRAW | CS_VREDRAW; //窗口风格
wcex.lpfnWndproc = (WNDPROC)WndProc; //窗口过程，处理消息响应
wcex.cbClsExtra = 0;
wcex.cbWndExtra = 0;
wcex.hInstance = hInstance; //程序实例
wcex.hIcon = 0; //图标
wcex.hCursor = LoadCursor(NULL,IDC_ARROW); //光标
wcex.hbrBackground = (HBRUSH)(COLOR_WINDOW+1); //背景色
wcex.lpszMenuName = NULL; //菜单资源
wcex.lpszClassName = "Your Class Name"; //窗口类名
wcex.hIconSm = NULL; //小图标
RegisterClassEx(&wcex); //注册窗口类
```

在此之后，就可以使用函数 CreateWindow 创建窗口了。不过，这个函数还有 11 个参数，第 1 个 参数就是：

```
wcex.lpszClassName = "Your Class Name";
```

只有用注册过的窗口类名才可以创建窗体。不过，用户一般情况下不用担心这些事情，因为，MFC 已经做了其中的大部分事情。如：

(1) 在 3 个应用程序框架里，主要的窗口都已经被创建了，可以直接使用 C++对象；

(2) 在资源编辑器里，控件窗口也被设计好了，您可以用 Class Wizard 为控件窗口连 接 C++对象。

(3) 如果是动态创建控件，您只需要用 Create 等函数来创建窗口，这一过程中，MFC 提 供了方法来判断窗口类是否注册。若未注册，则先注册，再创建窗口（您完全可以不了解这 些内容）。

3.2.3 Windows 编程基础

1.环境变量

和类 Unix 系统一样，Windows 系统也有环境变量，分为系统环境变量和用户环境变量 两种。

一般来说，系统环境变量影响所有的用户和程序，但更改系统环境变量后，对已经启 动的服务程序并不起作用，因为此服务程序并不知道环境变量已发生改变，需要重新启动操 作系统。

在命令提示符下，运行 set 可以看到当前所有的环境变量。同时， set 也可以更改环境 变量的值，但仅对当前的命令提示符下启动的程序起作用。

Windows 也提供 GUI 工具来修改环境变量，做法是单击【开始】菜单， 或者在【我的电脑】图标上单击鼠标右键，在弹出的菜单中选择【属性】命令，弹出【系统属性】对话框。切换到【高级】选项页，单击下方的【环境变量】按钮，弹出【环境变量】对话框，可以查

看或修改增加环境变量，此时的界面如图 3-2 所示。



图 3-2 环境变量的查看设置对话框

环境变量 PATH 的功能和 Unix 下的 PATH 加上 LD_LIBRARY_PATH 是一样的，同时影响可执行文件和动态链接库。执行命令时， Windows 会根据是否含路径，是否是 cmd 内部命令， 当前路径下是否有可执行扩展名的文件，然后再在 PATH 环境变量（先系统的再到用户的） 路径里查找有无可执行扩展名的文件。如果匹配，则执行。对于动态链接库，则是查找是否 包含路径，系统的 system32（32 位 OS 情况下）路径，然后是 PATH 环境变量（先系统的再 到用户的）路径里查找动态链接库的文件。

其他需要传递的信息对 C/C++程序员影响比较小，因为传递信息的方式已经被存取注册 表这种方式取代。

2.注册表

注册表可以看作是一个系统的数据库，大量系统及用户的数据在此存放，尤其是配置数据及运行状态数据。这些数据有字符、数字及二进制数组等类型。通过以 Reg 开头的一系列函数，可以存取注册表数据。

需要注意的是，注册表中的数据是透明的，其他程序及 regedit、regedt32 等系统工 具，也可以存取这些数据，这些数据 并不是被某个用户所独占的。

3.开机自动运行程序

通常我们开发的许多处理程序，往往是无人职守的开机自动运行的。

有许多种方式，可以让程序开机自动运行。通常可以在启动组里面，建立要运行程序的快捷方式。另外，可以放到全局或用户的启动注册表里面。全局的注册表项位于 “我的 电脑\HEKY_LOCAL_MACHINE\software\microsoft\Windows\currentversion\run”里面，用户注册表项在“我的电脑\HKEY_CURRENT_USER\software\microsoft\Windows\currentversion\run”里面。图 3-3 是用 regedit.exe 程序打开的注册表里面的全局启动程序数据。可以用工具或注册表的 API函数修改数据。

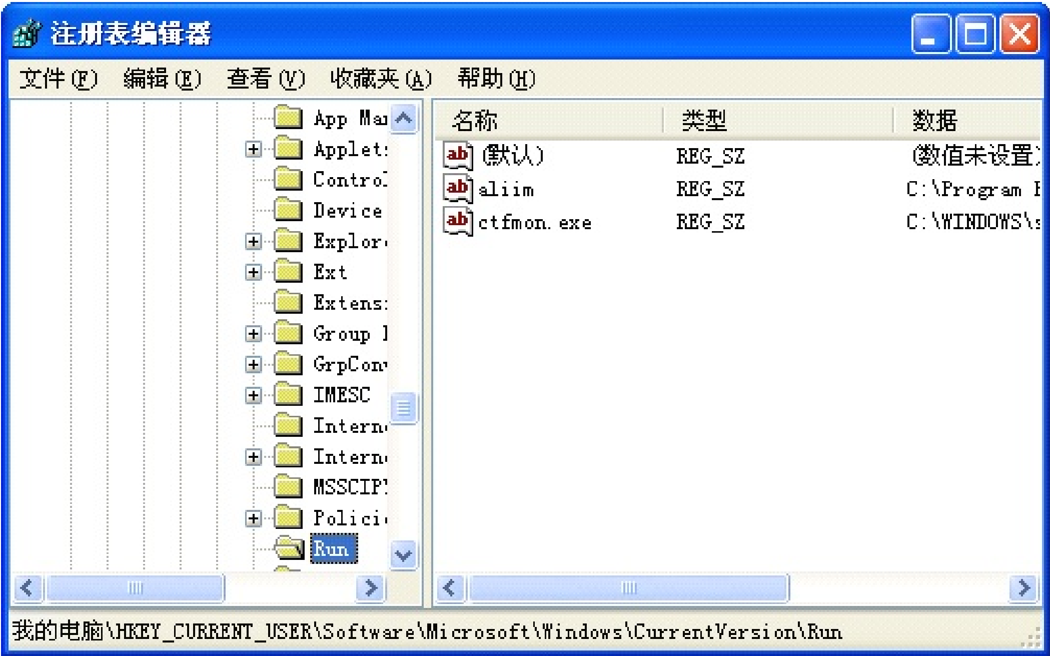


图 3-3 注册表中的全局自动运行程序数据

上述方法，有一个前提，就是用户必须能够自动登录系统。这可以通过修改注册表来 解决，方法是在注册表项 HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\WindowsNT\CurrentVersion\Winlogon 下面，增 加 DefaultUserName、DefaultPassword、AutoAdminLogon 三项数据，分别填入用户名和 密码，及数字 1，其中，DefaultUserName， DefaultPassword 的类型是字符串，AutoAdminLogon 的类型是数字。这样做虽然可以实现目标，但是存在安全隐患，就是你的用户密码以明文保存在注册表里面，这通常是非常危险的。

还可以采用其他方法来规避安全隐患。采用任务计划是一种方法，把计划的属性设为 在系统启动时启动，把计划的命令设置为要自动启动的程序。

其实最理想的方法是，把程序写成服务程序，设定此程序为服务，启动方式设为自动 启动。当然也可以使用 sc 命令，把普通的程序设为服务并设置为自动启动。

需要注意的是，如果自动运行的程序导致了系统错误，则很难进行跟踪调试，定位故障。

4.服务程序

服务程序是一种特殊机制的程序，用户不能直接运行或停止服务程序的可执行文件， 而是通过控制服务的启动停止来启动和退出。这些程序启动可以不需要用户登录就自动运 行，可以不需要 GUI 界面，非常适合业务处理类型程序的运行方式。

在控制面板里的服务管理面板中可以查看、启动和停止服务程序。

在 DOS 命令提示符环境下，可以用 net start 命令查看启动的服务列表；用“net start 服务名”命令启动服务；用“net stop 服务名”停止服务。

命令 sc 可以用来增加、删除、设置、启动或停止服务。

5.使用 Visual C++

Windows 编程是很复杂的事情。我们使用微软提供的 Develop Studio 中的 Visual C++ IDE，它能够处理编译和连接，提供帮助参考，自动生成程序框架代码，并且提供一种可视 化的设计环境。

在开始使用 Visual Studio 之前，需要掌握一些相应的知识和技能。最为重要的是要 熟练掌握在线帮助－MSDN 的使用方法。

然后是最重要的组合键 Ctrl+W。这个组合键能够调出 Class Wizard。Class Wizard 能 够为程序员在项目中加入代码，能够处理连接函数与 Windows 发送的消息的代码。

我们也会经常用到 Resource 视图面板。它可以用于设计 GUI。可以在对话框或其他框 架上放置窗口部件。通常情况下，通过若干次鼠标的点击，就可以完成界面的布局。然后再 使用 Class Wizard，完成类和对象的添加。剩下的工作主要是完成 Class Wizard 生成的函 数，并处理消息等。

这里不准备详细讨论 Visual C++的用法，并且从 VC6.0 到现在的 VC2008，每个版本都 拥有大量的使用者，而它们的操作方式也 不尽相同，请读者朋友在使用时阅读相关的书籍并 用心体会。

6.使用 MSDN

笔者最初学用 Visual C++时，经常会因为不知道某些类和方法如何使用而去问别人。 有一次，因为要问一个问题，把一位老鸟请到我的机器上，那位同事顺手就在我的计算机上 找 MSDN，找了半天没有找到，经询问我得知并没有安装时，他非常惊讶的说：没有装 MSDN 怎么搞开发啊？

这是真实的故事，从中我们可以看出 MSDN 在 Windows 平台的软件开发的重要地位，但 需要澄清的一个错误观念是 MSDN 并不是专门与 Visual C++配套使用的，它的内容涵盖了在 Windows 上开发所需的方方面面的内容。所以可以这样说，只要是在 Windows 上进行应用开 发，就需要使用 MSDN。下面我们就来了解和学用一下 MSDN。

(1) MSDN 简介

MSDN 是 Microsoft Software Developer Network 的简称，是微软针对开发者的帮助网络，可以在 http://msdn.microsoft.com 看到有关的详细介绍。MSDN 可以单独购买订阅，也可以在购买 Visual Studio 套件时得到。

(2) MSDN 的使用。MSDN 的安装比较简单，只需要按照向导提示安装即可完成。安装完 毕后，可以在 Windows 的开始菜单里找到【Microsoft Developer Network】→【MSDN Library for Visual Studio 2005】这一项（因为笔者安装的是这个版本，你的可能会有所 不同），通过这个菜单就可以打开 MSDN 了，也可以在 Visual Studio 中按下 F1 键打开 MSDN。打开后的 MSDN 主界面如图 3-4 所示。



图 3-4 MSDN 主界面

MSDN 菜单操作方式和一般的 Windows 帮助文件的样式并无不同。

3.3 Linux 编程基础

3.3.1 你必须掌握的技能

根据笔者的经验，要想在 Linux 上能够”无障碍”的用 Qt 编程，掌握必要的 Linux 技 能是必需的。以下是笔者列出的一些技能，供参考：

- 了解各个发行版的特点，能够根据需求，挑选和安装适合自己的发行版；
- 掌握常见的软件包管理工具的使用，包括 GUI 工具和编译命令，能够熟练安装软件包
- 熟悉 Linux 文件系统结构，能够熟练使用文件系统操作命令，配置文件的权限
- 了解 X Window 系统的原理和组成结构，必要时可以自行配置
- 能够熟练使用常见的集成式桌面环境，如 KDE 和 GNOME
- 熟悉 Linux 系统的帐号管理，能够熟练管理用户与组的帐号，并使用常用命令
- 能够熟练配置网络连接，包括局域网和 Internet
- 熟悉 Shell，了解 Shell 的种类，熟悉与 Shell 有关的配置文件，并能够根据需要修改
- 熟悉 Linux 文件压缩的方法，掌握文件压缩的常见命令
- 能够熟练使用 man 命令执行在线查询和获取帮助，能够熟练使用查找命令
- 能够熟练掌握各个发行版上基础编程环境的配置，主要包括 GCC、GDB 等
- 能够熟练使用一种代码编辑器，如 vi、emacs 等，推荐 vi
- 了解 Linux 系统下打包软件的常见方法

在下面的各节中，笔者将结合编程实际和本书内容的需要，有选择的向大家介绍这些技能，更为详尽的说明，请参考相关的 Linux 书籍。

3.3.2 文件系统管理

文件系统操作系统实际用来保存和管理各种文件的方法。由于每种操作系统支持的文件系统数量和类型都不同，因此在了解系统运行前，必须对文件系统的结构有所了解。尤其在 Linux 系统中，任何软硬件都被视为文件，所以这一部分的内容就尤为重要。

1.Linux 文件系统架构

操作系统中的文件系统（File System）可说是最基本的架构，因为几乎所有与用户、 应用程序或安全性模型相互通信的方法，都与文件保存的类型息息相关。

(1) 文件类型 简单来说，文件系统可以分为两种类型。

- 共享与非共享文件：共享文件是指允许其他主机访问的文件，而非共享文件则只供 本机使用
- 可变与固定文件：可变文件是指不需要通过系统管理员修改，即可自动更改内容的 文件，例如数据库文件；而固定文件则是指内容不会自动更改的文件，例如一般的文件或二 进制文件。

Linux 文件系统采用层次式的树状目录结构，此结构的最上层是根目录 “/”，然后在 根目录下再建立其他的目录。虽然目录的名称可以自定义，但是某些特殊的目录名称包含重 要的功能，所以不可随意将它们改名，否则会造成系统错误。

因为 Linux 允许不同的厂商及个人修改其操作系统，所以常会造成目录名称不统一的 情况，有鉴于此，目前有一套规范文件目录的命名及存放标准，它被称为 Filesystem Hierarchy Standard(FHS)，这也是大多数 Linux 发行版遵循的标准，如果需要详细的说 明，请参考以下网站说明：http://proton.pathname.com/fhs/。

(2) Linux 默认目录

在安装 Linux 时，系统会建立一些默认的目录， 并且每个目录都有其特殊功能，如图 3-5 所示。

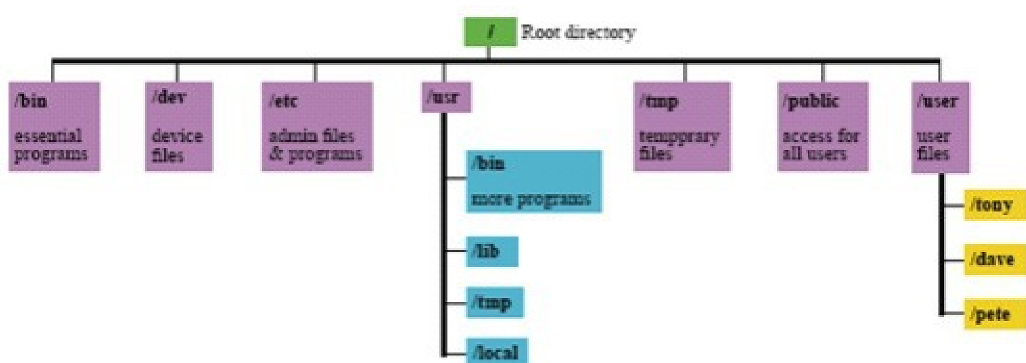


图 3-5 Unix/Linux 文件系统结构图

以下是这些目录的说明：

- / : Linux 文件系统的最顶层根目录
- /bin : Binary 的 缩写，存放用户的可执行程序，例如 ls、cp 和 mv 等，也包含其 他的 Shell，例如 Bash 和 csh
- /boot : 操作系统启动时需要的文件，例如 vmlinuz 和 initrd.imp。这些文件如果 损坏，常会导致系统无法正常启动，因此最好不要任意改动
- /dev : 接口设备文件目录，例如 hda 表示第一块 IDE 硬盘
- /etc : 有关系统设置与管理的文件，例如 password。目前保存在 etc/目录中的文 件都不是二进制文件，之前在此目录中的二进制文件都已移到 /sbin 或/bin 目录
- etc/X11 : X Window 系统的设置目录
- /home : 一般用户的主目录或 FTP 站台目录
- /lib : 仅包含执行/bin 和/sbin 目录中的二进制文件时所需的共享类库 （ shared library）
- /mnt : 各项设备的文件挂载（Mount）点，例如光盘的默认挂载点是/mnt/cdrom，而/mnt/floppy 是软驱的默认挂载点
- /opt : 这个目录通常提供一个空间，供较大型且固定的应用程序软件包保存文件使 用，这样可以避免将文件分散到整个文件系统
- /proc : 目前系统内核与程序执行的信息，它和用ps命令看到的内容相同
- /root : root 管理员主目录，其他用户的主目录都位于 /home 目录中

- /sbin：是 System Binary 的缩写，此目录存放的是系统启动时需要执行的程序， 比如 swapon
- /tmp：Tempoary 的缩写，它是用来存放临时文件的目录
- /usr：存放用户使用的系统命令，以及应用程序等信息
- /usr/bin：存放用户可执行程序，例如 finger 和 mdir 等
- /usr/include：保存供 C 语言加载的头文件
- /usr/include/X11：保存供 X Window 程序加载的头文件
- /usr/lib：用户类库
- /usr/lib/X11：X11 的类库
- /usr/local：提供自行安装的应用程序位置
- /usr/sbin：存放非经常性使用的程序，例如 showmount
- /usr/src：保存程序的原始文件
- /usr/X11R6/bin：存放 X Window 系统的执行程序
- /var：Variable 的缩写，具有可变性质的相关程序目录，例如 log、spool 和 named 等

(3) Linux 文件名称

Linux 中的文件名称最长可允许 256 个字符，而这些字符可用 A~Z、0~9、_、-等字符 来命名。

与其他操作系统相比，比如 DOS 和 Windows，Linux 最大的不同是，它并没有扩展名的 概念，也就是说文件的名称和该文件的类型并没有直接的关联，例如 sample.txt 可能是执 行文件，sample.exe 也有可能是文本文件，甚至可以不用扩展名。

另一个特性是文件名称区分大小写（Case Sensitive），这也是习惯 DOS 和 Windows 平台用户最难适应的一点。但所有的 Unix（包括 Linux）都遵循这条原则，例如 sample.txt、Sample.txt、SAMPLE.txt、sampLE.txt 在 Linux 上代表不同的文件，但在 DOS 和 Windows 平台上却是指同一个文件。

(4) Ext3 文件系统

Linux 可说是一种兼容性很高的操作系统。它可以支持的文件系统有很多，例如 msdos、ntfs、vfat、iso9660、minix 等等，也就是说它可以和其他许多不同的文件或操作 系统同时存在于硬盘中，这也是 Linux 足以傲视其他操作系统的地方之一。

与 micorsoft Windows 操作系统不同的是，Linux 并没有分区的概念，也就是说，它不 会将文件的保存位置指定为磁盘驱动器 C、D、E 等等，而是使用树状的 ext3fs（Third Extended File System）作为主要的文件系统（有时也简称为 ext3）。

ext3fs 是大多数 Linux 默认的文件系统，它主要是在原有的 ext2fs 系统上增加了日志 功能，并且在数据的有效性、完整性、访问速度方面有了大幅提高。

(5) Linux 文件路径

在此我们要介绍 Linux 文件路径表示法。因为在 Linux 的世界中没有微软产品的磁盘 驱动器的概念，也就是说没有所谓的磁盘驱动器 C、D 等表示法，它是利用目录与子目录的 层次式（Hierarchical）概念来表示文件保存位置的。

所以你需要先建立起这个新概念，否则可能会因为先前的概念而产生学习 Linux 的障 碍。一般来说，Linux 的文件路径分为两类：绝对路径和相对路径。

所谓绝对路径，就是指以根目录（/）为起始点来表示的路径，例如“/etc/ppp/peers/isdn/avm”就是绝对路径，也就是说，如果一个路径的表示法是以根目 录（/）开头，那就属于绝对路径；如果不是以根目录开头，就称为相对路径。

相对路径是指从当前的目录算起。我们以刚才的绝对路径为例，如果现在工作的目录是/etc/ppp，那么用相对路径来表示就是“./peers/isdn/avm”（也可以省略为 peers/isdn/avm）。

因为系统会自动在此路径前加上当前的工作目录位置，所以适当的使用相对路径可以 节省输入时间，并避免错误产生。

3.3.3 X Window 系统

许多人在抗拒学习 Linux 时，最大的借口之一就是命令和参数多如牛毛，特别是惯用 微软 Windows 操作系统的人更难接受。不过目前在 Linux 上已经大幅进步的图形界面功 能，使得 X Window 系统成为学习 Linux 时的另一种选择。

1.X Window 系统基础

X Window 系统是一种图形化的操作环境，它可以在 Unix 和 Linux 操作系统上提供 GUI（Graphical User Interface，图形用户界面）操作界面，同时它也有以下几种不同的名称：X、X 视窗、X11、X11R7。

在此特别提醒一点，虽然以上名称都可用来表示 X Window 系统，但却不可称为“Windows”，因为 Windows 一词已被 Microsoft 公司注册，所以切勿使用 Windows，以免 侵犯该公司的注册商标。

(1) X Window 系统的起源

在 X Window 系统出现前，其实已经有很多公司在发展 Unix 用户图形界面，但由于每 家公司开发的规范不一，因此在兼容性方面表现不佳，这种情形直到 X Window 系统推出后 才得以解决。

X Window 系统起源于 1984 年的雅典娜（Project Athena），它是由麻省理工学院（MIT）与 Digital Equipment 公司合作开发的图形界面系统，因为它以斯坦福大学的 W Window 系统为基础，所以命名为 X Window 系统（因为字母 X 位于 W 之后）。

为了确保 X Window 系统的持续发展，MIT 于 1988 年成立 X Consortium，它由 X Window 系统的主要设计者 Robert W.Scheifler 负责。之后由于 X Window 系统引起许多公司的兴趣，所以新版的 X Window 系统不断问世。

(2) X Window 系统组成

X Window 系统采用客户端一服务器架构，其中主要的组件为：X Server 和 X Client。 前者负责驱动显示卡和各种图形的显示，同时也会驱动其他输入设备，使客户端可以通过这 些输入界面与应用程序通信。而后者指实际执行的应用程序，它会向 X Server 提出服务请 求，以得到响应的显示画面。

除了 X Server 和 X Client 之外，在 X Window 系统中还包含其它组件，如图所示，以 下是这些组件的说明：

X Protocol

介于 X Server 和 X Client 间用于沟通的通信协议。因为基本的 X Window 系统并不提 供用户界面，比如按钮和菜单等组件，所以必须依靠 X Protocol 中的程序来提供这类功 能，否则单纯的 X Window 系统无法满足客户端的要求。

X Library

最底层的程序界面，它的主要功能是存取 X Protocol 服务，这对于图形程序编写非常 重要，常见的 X Library 有：Xlib、Motif、Qt 和 GTK+等。

X Toolkit

包含在 X Library 中的应用程序发展工具，它提供了 X Window 设计时需要的基本函 数，避免了程序开发时必须自行设计所有组件的不便，例如滚动条和功能钮。这些组件也称 为 widgets。目前 X Toolkit 的种类很多，较常见的有：Motif Development Toolkit、 OpenLook Toolkit、GTK+、TCL/TK、XForms 和 X Toolkit（Xt）等。

(3) X Window 系统的特点

X Window 系统的特点可以归纳如下：

- 图形化界面

  X Window 系统是在 Linux 中唯一的图形界面系统，但是可以搭配多套窗口管理程序使用，是比 Windows 产品更具有弹性设计。如果希望修改某些窗口管理程序的内容，可以用 它的源代码进行修改。

- 支持多种应用程序

  目前在 X Window 系统中可使用的应用程序越来越多，文字处理、多媒体、图形图像、游戏软件、因特网、甚至系统管理工具，都有免费的图形化工具可供使用。这除了有助于消 除用户对于文字界面的陌生感，还可以使其可能逐步取代 Windows 而成为个人工作

站的选 择。

- 弹性设计

  因为在 X Window 系统的设计中，X Server 只负责基本的显示及终端的控制，而其余的部分都是由 X Client 处理，所以这种设计不受操作系统的限制，突同的操作系统都可以使用 X Server。特别是在 Unix 的多任务环境中，更能发挥其优异的特性。

- 客户端一服务器架构

  X Window 系统采用客户端一服务器架构，它将系统显示功能与应用程序分别用 X Server（X11R7）和 X Client 来执行，这种架构最大的优势是，执行程序（ X Client）和显示结果（X Server）的主机可以是不同的两部计算机。

举例来说，如果希望在主机上使用 firefox 来浏览网页，但是主机尚未安装 firefox， 而网络的某个工作站上装有 firefox，此时即可连接此工作站并执行 firefox，再将结果返 回本地主机。

在以上的例子中，本地主机担任 X Server 的角色，它只负责显示结果，而远程装有 firefox 的 Linux 工作站则是 X Client，它负责执行程序，在网络间负责传递两端信息的 机制则为 X Protocol。

通过 X Protocol 这个媒介，所有的主机都能使用此软件，而不需要在每台计算机上安 装相同的软件。因此建议将大型且需要高速运算的软件交由能力较强的工作站来处理，然后 再用网络显示执行结果。

2.X11R7

X11R7 是目前多数主流 Linux 发行版使用的 X Server，它发布于 2005 年 12 月，与之 前的 X11R6.9 相比，采用了模块化的设计。

(1) X11R7 重要目录

与 X11R7 有关的软件，大多放在/usr 及其子目录中，以下是较为重要的目录的说明。

- /usr/bin：存放 X Server 和不同的 X Client
- /usr/include：开发 X Client 和图形所需的文件路径
- /usr/lib：X Server 和 X Client 所需的类库目录
- /usr/lib/X11：保存多项资源，例如字体和文件等
- /usr/lib/xorg/moudles：包含驱动程序与多种 X Server 模块
- /usr/X11/man：保存 X11 程序编写时的手册说明

(2) /etc/X11/xorg.conf 文件

在安装 Linux 时如果没有设置 X Window 系统，之后必须先设置鼠标、键盘、显示器以 及显示卡等，才能成功启用 X Window 系统，而这些设置都记录在/etc/X11/xorg.conf 文件 中，由此可见这个文件的重要性。

这个文件由数个 Section/EndSection 的区块组成，每个区块的格式如下：

```
Section “Section 名称”
选项名称 “选项值”
选项名称 “选项值”
选项名称 “选项值”
......
EndSection
```

在/etc/X11/xorg.conf 文件中，有多个区块是非常重要的，包括 ServerLayout、

Files、Moudle、InputDevice、Monitor、Device、Screen 和 DRI 等。从字名上不难看出它们的用途，详细的设置请参阅相应的 Linux 发行版手册，这里不再赘述。

3.集成式桌面环境

惯用 Windows 操作系统的用户可能没有想过，为什么每个人的右键菜单、开始菜单、 所有程序以及其它的系统设置都是如出一辙？能不能随着用户的喜好进行修改呢？

的确，Windows 系统使用单一的图形界面，并且用户无法改变它。而 Linux 在这方面就 具有较大的弹性，用户可以依其喜好随时改变图形界面，这就是所谓的集成式桌面环境。在 我们使用 Qt 开发的时候，经常用到的两种桌面环境是 GNOME 和 KDE。

(1) GNOME

GNOME 是 GNU Network Object Model Environment 的缩写，它属于 GNU（GNU‘s Not Unix）项目的一部分，这个项目始于 1984 年，目的是发展一个完全免费的类 Unix 操作系统。GNOME 目前是许多 Linux 发行版默认的桌面环境。

除了包含功能强大的组件外，GNOME 也具有灵活的可配置性，所以可以根据个人的喜好 和习惯设置桌面环境。

(2) KDE

KDE（K Desktop Environment）是目前 Linux 两大集成式桌面环境之一，它与 1996 年 10 月由 Lyx 的原创者 Matthias Ettich 发起，从而促成了 KDE 计划的产生。

与 GNOME 最大的不同是，KDE 原本是使用 QPL（商业版权）发展的，而 GNOME 则是遵循 GNU 协议的，因此在刚开始时，KDE 的推广遇到了一定的障碍。直到 Qt 决定使用 GPL 协议 后，KDE 的发展又进入了快车道。

Linux 桌面环境的使用相对比较容易，可以参阅系统提供的帮助。

3.3.4 常用命令

下面我们介绍一些常用的命令，它们是经常会被用到的，需要熟练掌握。这里并不展 开对各个命令的详细说明，使用时请参考相关书籍，也可以使用 man 命令来获得帮助。

表 3-4 Linux 常用命令说明

man	获得联机帮助，是类 **Unix** 用户的在线帮助手册
cd	切换当前路径命令
pwd	显示当前路径
ls	显示目录下面的文件和子目录情况
chmod	变更文件和目录的属性
mkdir	建立目录
rm	删除目录或文件
su	切换用户登录到 shell，常见从一般用户到 root 用户或者相反顺序
exec	执行程序，并且不返回到当前 shell
ldd	查看应用程序使用的动态库
nm	查看程序或库的调试信息
objdump	查看程序或库的信息
env	查看环境变量
grep	从文件中查找字符串
find	查找文件
which	查找命令的可执行文件
uname	查看操作系统版本
ps	查看进程信息
top	查看系统资源信息
vmstat	查看系统虚拟机各资源信息
vi/vim	使用 vi/vim 编辑器
make	处理工程文件，生成可执行文件或库或其他资源文件
gdb/dbx	调试工具命令
strace/ltrace	跟踪程序调用的系统函数情况
file	查看文件的格式
fuser	查看进程使用了哪些文件

3.3.5 Shell 应用

Shell 在操作系统中的作用就如同是翻译员，它在用户和操作系统之间传递信息。如果 少了它的运行，那么用户和操作系统之间将被完全阻隔而无法沟通。因此，了解 Shell 是 深入 Linux 核心、学习 Linux 上 Qt 开发的基本功课之一。

1.Shell 基础

Shell－如同其名字一样，它就像是一个壳，而这个壳介于用户和操作系统（ Kernel）中间，负责将用户的命令解释为操作系统可以接受的低级语言，同时将操作系统的响应信息 以用户了解的方式来显示，这样可以避免用户执行不当的命令而对系统产生损害。图描述的是 Shell 这种角色。

每个用户在登录 Linux 后，系统会出现不同的提示符号，如#、$或~等，之后就可以输 入需要的命令。如果命令正确，系统就可按照命令的要求来执行，直到用户注销系统为止。 在登录到注销期间，用户输入的每个命令都会经过解释并执行，而这个负责的机制就是 Shell。

(1) 命令的类型

一般用户的命令可分为两大类：程序和 Shell 内置命令。如果该命令为程序类型，那 么 Shell 会找出该程序，然后将控制权交给内核，并由内核负责执行该程序；而在内核将 程序执行完毕后，再将控制权交给 Shell。但如果是 Shell 内置命令，则由 Shell 直接响 应，因此速度较快。

要判断一个命令属于 Shell 的内置命令还是程序，可以用 find 命令来判断：如果 find 命令没有任何响应，则表示该命令为 Shell 内置命令；如果显示查找的结果，则该命令为 程序。

其实 Shell 的概念并不只存在与 Linux 系统，在其他的操作系统上也有，只不过名称 不同，如 DOS 中的 command.com 和 Microsoft Windows 的 GUI（Graphical User Interface）。

但是 Linux 操作系统对于 Shell 极具灵活性的使用，是其他操作系统望尘莫及的。在 Linux 中可以使用的 Shell 很多，并且可以随意更换不同的 Shell。

(2) Shell 的种类

Linux 支持的 Shell 都记录在/etc/shells 文件中，我们可以使用 cat 命令来查看支持 的 Shell。

```
[wd@localhost ~]$ cat /etc/shells

/bin/sh
/bin/bash
/sbin/nologin
/bin/tcsh
/bin/csh
/bin/ksh
```

虽然每种 Unix/Linux 系统可以兼容的 Shell 有很多，但是使用较广的只有三种： Bourne Shell（sh）、C Shell（csh）以及 Korn Shell（ksh）。

每种 Shell 的命令名称和登录时出现的提示符号不尽相同，表 3-5 是个简单的说明。

表 3-5 常见 Shell 的说明

Shell 名称	命令名称	登录符号
Bourne	/bin/sh	$
C	/bin/csh	%
Korn	ksh	$

I. Bourne Shell

Bourne Shell 是最早被大量使用和标准化的 Shell，几乎所有的 Unix/Linux 都支持。 由于 Bourne Shell 在执行效率上优于其他的 Shell，所以它是大多数 Unix 系统的默认Shell。但是它并不支持别名（aliases）与历史记录（history）等功能，同时在作业控制（Job Control）上的功能也比较简单，所以整体而言，在目前的系统环境下已略显不足。

II. C Shell

C Shell 的语法与 C 语言类似，它因此而得名。C Shell 已经是 Unix 类操作系统的重 要组成部分之一。

C Shell 的特点在于易于使用以及交互性强，目前 BSD 版的 UNIX 大多以 C Shell 作为 默认的 Shell。

III. Korn Shell

Korn Shell 兼具 Bourne Shell 和 C Shell 的优点，并且语法与 Bourne Shell 兼容， 但它出现的较晚，在一些新版的 Linux 发行版如 Ubuntu 中才有支持。

(3) 更改 Shell

因为 Linux 可以支持的 Shell 有很多，所以大家可以根据个人的习惯选择使用不同的 Shell。要查看当前使用的 Shell 或系统默认的 Shell，最简单的方式就是使用 echo 命令来 查询系统的 Shell 环境变量，命令用法如下：

```
[wd@localhost ~]$ echo $SHELL

/bin/bash #当前使用的 Shell 为 bash
```

或者输入：

```
[wd@localhost ~]$ echo ${SHELL}

/bin/bash
```

但是以上的方法只能显示用户登录时使用的 Shell，而无法显示出更换过的 Shell。如

果要更改使用的 Shell。只要执行该 Shell 程序名称，即可切换到不同的 Shell。下面是更 改 Shell 的方法：

```
[wd@localhost ~]$ sudo sh
#执行 sh
$ sudo bash
#执行 bash
```

在一般情况下，执行 exit 命令会立即注销系统。但如果从默认的 Shell 切换到其他的Shell，则不论切换的次数有多少，在切换后使用 exit 命令都不会注销系统，而只会跳离 当前的 Shell，并回到上一层的 Shell。

以上切换 Shell 的方法虽然简单，但它只是暂时的改变，待用户注销后登录，又会回 到系统默认的 Shell。

要解决上述问题，可以使用 chsh 命令（Change Shell），它的使用方法很简单。下面 是将用户默认的 Shell 改为 csh 的例子：

```
[wd@localhost ~]$ sudo chsh
正在更改 root 的 Shell
请输入新值，或直接敲回车键以使用默认值
登录 Shell [/bin/bash]: /bin/csh
# Shell 更改成功
```

注意，在使用 chsh 命令更改用户默认的 Shell 后，要重新登录才会更改。

2.环境变量

所谓的“环境变量”（Environment Variables），是指 Shell 中用来保存系统信息的 变量，这些变量可供 Shell 中执行的程序使用。不同的 Shell 会有不同的环境变量名称和 环境变量值的设置方法。

在 bash 中要显示环境变量名称及环境变量值，可以使用 set 命令。

3.Shell 配置文件与 Shell Script

在上一小节里，我们介绍了环境变量的内容，并了解了如何自定义变量名称或修改预 先定义的变量。而除了可以使用命令来执行变量的设置外，也可以通过一些 Shell 配置文 件来设置。在用户登录时，系统会检查这些配置文件，以便设置环境。

本小节我们将列出所有与 Shell 有关的配置文件名称，并且说明每个文件的功能。

(1) /etc/profile

这是系统最主要的 Shell 配置文件，也是用户登录时系统最先检查的文件。系统最重 要的环境变量都定义在此，其中包括 PATH、USER、LOGNAME、MAIL、HOSTNAME、HISTSIZE 和 INPUTRC 等。

除此之外，这个文件也定义了 ulimit，它的功能是限制每个 Shell 所能执行的程序数 目，以免造成系统资源的过度消耗。而在文件的最后，它会检查并执行

```
/etc/profile.d/*.sh 的 Script。
```

(2) ~/.bash_profile

这个文件是每位用户的 BASH 环境配置文件，它存在于用户的主目录中。当系统执行 /etc/profile 后，就会接着读取此文件内的设置值。

在此文件中，会定义 USERNAME、BASH_ENV 和 PATH 等环境变量。但是此处的 PATH 除了 包含系统的$PATH 变量外，还另外加入了用户的 bin 目录路径，而 BASH_ENV 变量则指出接 下来系统要检查的文件名称。

(3) ~/.bashrc

接下来系统会检查~/.bashrc 文件，这个文件和前两个文件（/etc/profile 和 ~/.bash_profile）最大的不同是，每次执行 bash 时，~/.bashrc 都会被再次读取，也就是 说变量会再次被设置；而/etc/profile 和~/.bash_profile 只有在登录时才进行读取。

就是因为经常被重新读取，所以~/.bashrc 文件只用来定义一些终端设置及 Shell 提示 符号等，而不用来定义环境变量。

举例来说，如果远程的终端窗口（例如由微软平台以 Telnet 进行登录）无法浏览超过 一页的信息或文件内容，可以在此文件中加入下面这行：

```
export TERM=vt100
```

~/.bashrc 文件中值得注意的一行是“. /etc/bashrc”，它利用一个小数点接着一个空格键再指向另外一个 Script，表示同时执行此 Script，并且采用 Script 的变量设置。

(4) ~/.bash_login

如果~/.bash_profile 文件不存在，则系统会转而读取这个文件内容。这是用户的登录 文件，每次用户登录系统时，bash 都会读取此文件，所以通常都会将登录后必须执行的命 令放在这个文件中。

(5) ~/.profile

如果~/.bash_profile 和~/.bash_login 两个文件都不存在，则会使用这个文件的设置 内容。它的功能与~/.bash_profile 完全相同。

(6) ~/.bash_logout

这个文件是 bash 在注销系统前读取的文件。通常这个文件只包含 clear 命令，也就是 先清除屏幕再注销。如果想在注销 Shell 前执行一些工作，例如清空缓冲区或执行备份， 都可以在此文件中设置。

(7) ~/.bash_history 这个文件中会记录用户曾经使用的命令历史，以供查阅。

3.3.6使用库程序

编程库是可以在多个软件项目中重用的代码集合。库是软件开发核心目标（代码重用）的一个经典例子。它们把常用的编程例程和实用程序代码收集到单独位置。例如，标准的 C 库包含数百个常用的例程，如输出函数 printf()和输入函数 getchar()，如果 每次创 建新程序都要重写它们很让人厌烦。但是，除了代码重用和程序员获得便利之外，库还提供大量已完成调试和良好测试过的实用程序代码，如用于网络编程、图像处理、数据操作和系统调用的例程。

在创建、维护以及管理编程库时，需要知道可用的工具。有两类库：静态的和共享的。

1.静态库

静态库是包含目标文件的专门格式文件，这些文件称为模块或成员，是可重用的、预 编译的代码。以特殊格式将它们与一张表格或映射图（把符号名链接到定义符号的成员）存 储起来。映射图可加快编译和链接。静态库通常用扩展名 .a 命名，.a 代表归档（archive）。

2.共享库 又被称作是动态库。与静态库类似的地方是，共享库也是文件，它包含其他目标文件或者指向其他目标文件的指针。称它们为共享库是因为在编译程序时，不 需要将它们包含的代码链接到程序。相反，动态的链接器 /装载器在运行时把共享库代码链接到程序中。

与静态库相比，共享库有几个优势。首先，它们需要的系统资源较少。因为共享库代 码没有编译成二进制代码，而是在运行时从单个位置动态的链接和装载，所以它们使用更少 的磁盘空间。它们使用的系统内存也较少，因为只需一次即可将它们装载到内存。最后，共 享库简化了代码和系统维护。修复 bug 或添加特性后，用户只需获得已更新的库并且安装 它即可。但对于静态库，必须重新编译使用该库的每个程序。

小贴士：如果想要构建一个在主机系统上不依靠任何内容的应用程序，或者在不确定开发 环境的地方使用特殊的命令时，使用静态库也有一定的优势。但是通常建议使用共享库。

3.常见的库命令

(1) nm 命令

nm 命令列出目标或二进制文件中所有已编码的符号。使用它可查看程序调用了什么函 数，或者查看库或目标文件是否提供了所需的函数。 nm 的语法如下：

```
nm [options] file
```

nm 列出存储在 file 中的符号，该 file 必须是静态库或归档文件，正如前面小节所描述的。options 控制 nm 的行为。符号类似与在代码中引用的函数、来自其他库的全局变量 等内容。必须跟踪找到程序所需的丢失符号时，可以使用 nm 命令作为一个工具。

表 3-6 介绍了有用的 nm 选项。

表 3-6 nm 命令行选项

选项	描述
-c	把符号名转换成用户级别名。这对使 C++函数名可读特别有用
-l	使用调试信息打印每个符号定义的行号，如果符号未定义，则重定位输入项
-s	在归档文件（.a）上使用时打印索引，该索引把符号名映射到定义符号的模块或成员上
-u	只把未定义的符号、外部定义的符号显示到正被检验的文件

下面是一个例子，它使用 nm 显示/usr/lib/libdl.a 中的一些符号：

```
$ nm /usr/lib/libdl.a | head
dlopen.o:
0 0 0 0 0 0 4 0 T __dlopen_check
U _dl_open
U _dlerror_run
0 0 0 0 0 0 4 0 W dlopen
0 0 0 0 0 0 0 0 t dlopen_doit
dlclose.o:
U _dl_close
```

(2) ar 命令

ar 可创建、修改或者提取档案。最常用于创建静态库（该库是包含一个或者多个目标 文件的文件）。ar 还创建并维护一个表，该表交叉引用符号名和定义符号名的成员。 Ar 命 令的句法如下：

```
ar {dmpqrtx} [options] [member] archive file [...]
```

ar 根据 file 中列出的文件创建名为 archive 的档案。至少要求使用 d、m、p、q、r、t 和 x 中的一个。通常使用 r。表 列出了最为常用的 ar 选项。

表 3-7 ar 命令行选项

选项	描述
-c	如果 archive 尚不存在，则通常发出禁止警告
-q	不检查是否有移位，把文件添加到 archive 末尾
-r	把文件插入 archive，代替任何现有成员中其名字与正增加的 文件名匹配的成员。新成员增加到档案末尾
-s	创建或者更新映射，此映射将符号链接到定义符号的成员

小贴士：给定一个用 ar 命令创建的档案，通过创建该档案的索引，可以加快档案的访问速度。ranlib 可精确的完成该任务，它在档案本身中存储该索引。 Ranlib 的句法是：

```
ranlib [-v | -V] file
```

这在 file 中生成符号映射，它等同于 ar -s file。

(3) ldd 命令

虽然 nm 可列出目标文件中定义的符号，但除非知道各个库定义了哪些函数，否则它没 什么太大用处。这就是 ldd 的工作了。它列出程序运行时所需的共享库。它的句法是：

```
ldd [options] file
```

ldd 打印 file 要求的共享库名。ldd 两个最有用的选项是-d（它报告任何缺失的函数）和-r（它报告缺失的函数和缺失的数据对象）。例如，下面的 ldd 报告 ftp 客户端 lftp（系统 上可能已安装它，也可能未安装）需要 13 个共享库：

```
$ ldd /usr/bin/lftp
liblftp-jobs.so.0 =&gt; /usr/lib/liblftp-jobs.so.0 (0x2aaf8000)
liblftp-tasks.so.0 =&gt; /usr/lib/liblftp-tasks.so.0 (0x2ab48000)
libreadline.so.5 =&gt; /usr/lib/libreadline.so.5 (0x2abbc000)
libutil.so.1 =&gt; /lib/libutil.so.1 (0x2ac08000)
libncurses.so.5 =&gt; /lib/libncurses.so.5 (0x2ac1c000)
libresolv.so.2 =&gt; /lib/libresolv.so.2 (0x2ac50000)
libdl.so.2 =&gt; /lib/libdl.so.2 (0x2ac78000)
libc.so.6 =&gt; /lib/libc.so.6 (0x2ac8c000)
libstdc++.so.6 =&gt; /usr/lib/libstdc++.so.6 (0x2ae24000)
libm.so.6 =&gt; /lib/libm.so.6 (0x2af44000)
libgcc_s.so.1 =&gt; /lib/libgcc_s.so.1 (0x2afdc000)
/lib/ld.so.1 (0x2aaa8000)
libtinfo.so.5 =&gt; /lib/libtinfo.so.5 (0x2b018000)
```

具体系统上的输出可能会有所不同。

(4) ldconfig 命令

ldconfig 确定共享库所需的运行时链接，它位于 /usr/lib 和/lib 中，在命令行上 libs 中指定，存储在/etc/ld.so.conf 中。它与动态链接器/装载器 ld.so 一起工作， 创建 并维护到系统上最新版本可用共享库的链接。它的句法如下：

```
ldconfig [options] [libs]
```

运行无参数的 ldconfig 只会更新缓存文件/etc/ld.so.cache。options 控制 ldconfig的行为。当 ldconfig 更新缓存时，-v 选项通知它是 verbose 的。-p 选项表示打印，但不 更新 ld.so 所知道的当前共享库列表。要查看更新缓存时 ldconfig 正在完成什么，-v 选项可以打印显示 ldconfig 已经找到的目录和系统链接。

4. 环境变量和配置文件

动态链接器/装载器 ld.so 使用很多的环境变量来自定义和控制其行为。这些变量包括：

```
$LD_LIBRARY_PATH
```

该变量包含一个用冒号分隔开的目录列表，运行时在该表中查找共享库。它类似于$PATH 环境变量。

```
$LD_PRELOAD
```

该变量是一个用空格分隔开的附加列表，其中包含用户指定在所有其他库之前装载的库。它用于有选择的覆盖其他共享库中的函数。

ld.so 还使用两个配置文件，其作用目的与那些环境变量平行：

```
/etc/ld.so.conf
```

包含一个目录列表，除了标准目录（/usr/lib 和/lib，以及 64 位架构系统上的/lib64），连接器/装载器还应该在该列表的目录内搜索共享库。

/etc/ld.so.preload 包含$LD_PRELOAD 环境变量基于磁盘的版本，它包括一个在执行程序之前装载的、用空 格分开的共享库列表。

可以借助$LD_PRELOAD 使用特定的版本来覆盖已安装的版本。在测试新的（或者不同 的）库版本，而又不想在系统上安装代替库时，该功能通常很有用。通常在创建程序时只使 用环境变量。在生产环境中不要依赖这些环境变量，因为在过去它们导致过安全问题，所以 可能无法控制变量的值。

3.3.3 使用 VI

一个常用的 Linux 开发工具箱中必须包含包括什么？基本上要包括一个编写代码的编 辑器、一个或者多个把源代码转化成二进制代码的编译器，以及一个跟踪无法避免的 bug 的调试器。多数人都有喜爱的编辑器，试图说服他们试用新的编辑器是很困难的事情。多数 编辑器

支持一组与编程有关的功能（可以肯定有些支持的功能会更多）。可以使用的编辑器 太多，由于篇幅有限无法意义介绍，但有一点是必须要声明的：至少需要一个编辑器。

只要使用 Linux，那么不使用文本编辑器几乎是不可能的。这是因为多数 Linux 配置文 件是纯文本文件，所以有时肯定需要进行手动修改。

如果正在使用 GUI，那么可以运行 gedit，编辑文本时使用它相当直观。还有一个简单 的文本编辑器 nano，可以从 shell 中运行它。但是多数 Linux Shell 用户会使用 vi 或 emacs 命令来编辑文本文件。与图形编辑器相比， vi 或 emacs 的优势在于可以在任何 shell、字符终端或基于字符的网络连接（例如使用 telnet 或 ssh）中使用它们，而无需使 用 GUI。它们都具有强大的功能，所以可以一直使用它们。

本节我们将提供一个简单的 vi 文本编辑器教程，使用它可在任意 shell 中手动编辑配 置文件。

1.运行 vi

通常情况下，运行 vi 可以打开特定的文件。例如，要打开 /tmp/test 文件，可输入下 面的命令：

```
$ vi /tmp/test
```

如果这是一个新文件，应该看到和下面类似的内容：

```
"/tmp/test" [New File]
```

顶部的框表示光标的位置，底部的行通知编辑情况（此处只是打开了一个新文件）。

在这两部分之间，波浪线（～）作为填充符，因为文件中还没有任何文本。现在这是令人害怕的部分：这里没有提示、菜单或图标告诉我们要做什么。不能只是从顶部开始输入。如果这样做，计算机就会发出蜂鸣声。所以有些人抱怨 Linux 并不友好。

(1) 首先需要了解的是不同的操作模式：命令或输入。

vi 编辑器始终启动到命令模式。在添加或修改文件中的文本前，必须输入命令（一个 或者两个字母加上一个可选的数字）告诉 vi 您想要做什么。大小写很重要，所以要按例子 所示精确的使用大写或小写字母！要进入输入模式，输入该输入命令。输入下面的命令开始 操作。

- a：添加命令。在它之后，可以从光标的右端开始输入文本。
- i：插入命令。在它之后，可以从光标的左端开始输入文本。 输入一些词句，然后按下 Enter 键。重复执行该操作数次，直到有几行文本为止。完成输入后，按下 Esc 键反回到命令模式。现在文件中有些文本了，试用下面的键或字母在文本中移动。记住使用 Esc 键，它始终可以回到命令模式。
- 方向键：在文件中上、下、左或右移动光标，一次一个字符。也可以使用退格键和 空格键分别向左和向右移动。如果喜欢将手指放在键盘上，可使用 h(左)、l(右)、j(下)、 或

k(上)来移动

- w：将光标移动到下一个单词的开头。
- b：将光标移动到前个单词的开头。
- $（零）：将光标移动到当前行的末尾。
- H：将光标移动到屏幕的左上角（屏幕上的第一行）。
- M：将光标移动到屏幕中间的第一个字符。
- L：将光标移动到屏幕的左下角（屏幕上的最后一行）。

(2) 其它编辑操作中唯一需要知道的是如何删除文本。 下面是一些删除文本用的命令。

- x：删除光标下的字符。
- X：删除光标前字符。
- dw：删除从当前字符开始直到当前单词末尾的所有字符。
- d$：删除从当前字符开始直到当前行末尾的所有字符。
- d0：删除从前一个字符开始直到当前行开头的所有字符。

(3) 要结束编辑，可使用下列击键保存和退出文件。

- ZZ：将当前修改保存到文件并退出 vi。
- :w：保存当前文件，但继续编辑。
- :wq：与 ZZ 相同。
- :q：退出当前文件。没有任何未保存的修改时该命令才会工作。
- :q!：退出当前文件，并且不保存对文件进行的修改。 小贴士:如果确实错误的修改了文件，那么 :q!命令是退出并且放弃修改的最好方法。文件会还原到最近修改的版本。所以如果只是使用 :w，有时可能会陷入困境。如果只想取消一些错误的编辑，按 u 键即可撤销修改。

(4) 常用技巧

现在已经学习了一些 vi 编辑命令。在后面会介绍更多的命令。这里先列出首次使用 vi 的一些提示。

- Esc：记住，Esc 用于回到命令模式（我曾看到有人按下键盘上的所有键来尝试退出 文件）。
- u：按 u 键可以撤销之前做的修改。连续按 u 键可以撤销更前面的修改。
- Ctrl+R：如果决定不再撤销前面的命令，可使用 Ctrl+R 进行恢复。本质上，这个 命令取消所做的撤销操作。
- Caps Lock：小心不要错按了 Caps Lock 键。处于大写状态时，在 vi 中输入的任何 内容都有不同含义。输入大写字母时不会出现警告，但事情却开始变得不可思议。
- :!命令：在 vi 中，可使用:!后跟命令名的方式来运行命令。例如，输入 :!date 查 看当前的时间和日期，输入:!pwd 查看当前目录，输入:!jobs 查看后台是否有任务正在运 行。命令运行完成时，按 Enter 键就可以返回继续编辑文件。甚至可以使用该技术从 vi 中 启动 shell（：！bash）、在该 shell 中运行几个命令，然后键入 exit 返回到 vi（我建议 转到 shell 前保存文件，防止回到 vi 后忘记保存）。

- —INSERT：处于插入模式时，INSERT 一词会出现在屏幕底部。
- Ctrl+G：如果忘记了正在编辑的内容，按下这些键可在屏幕底部显示正在编辑的文 件名和所在的行。它还显示文件的总行数、已浏览过的部分占该文件的百分比，以及光标所在的列号。这用来在下午停止工作一段时间后，帮助您确定编辑的位置。

2.搜索文本

要搜索文本在文件中下次出现的位置，可使用斜线（／）或问号（？）。在斜线或问号后面加上模式（字符串或文本）可分别向前或向后搜索该模式。搜索时也可以使用元字符。下面是一些例子。

- /hello：向前搜索单词 hello。
- ?goodbye：向后搜索单词 goodbye。
- /The.*foot：向前搜索包括单词 The，同时在 The 之后的某处有单词 foot 的行。
- ?[pP]rint：向后搜索 pring 或 Print。记住，Linux 中是区分大小写的，所以可使 用括号来搜索大小写不同的单词。

vi 编辑器最初基于 ex 编辑器，而 ex 编辑器不能完全在全屏幕模式下运行。但是它允 许运行命令，以便同时在一行或者多行中搜索和修改文本。输入冒号并且光标到达屏幕底部 时，实际上就处于 ex 模式下。下面的例子用 ex 命令搜索和修改文本（例如我选择搜索 Local 和 Remote，但也可以使用其它合适的单词）。

- :g/Local：搜索单词 Local，并且打印文件中它所出现的行（如果结果多于一个屏 幕，则以管道形式将输出定向到 more 命令）。
- :s/Local/s//Remote：在当前行上用 Remote 代替 Local。
- :g/Local/s//Remote：用 Remote 代替文件中每行第一次出现的 Local。
- :g/Local/s//Remote/g：用 Remote 代替文件中出现的所有 Local。
- :g/Local/s//Remote/gp:用 Remote 代替文件中出现所有的 Local，然后打印没一行 来查看进行的修改（如果输出多于一页，则以管道的形式将输出定向到 more 命令）。

3.使用命令和数字

在多数 vi 命令前都可以使用数字，这样命令就能够重复执行该指定数目的次数。这是 一次处理多行、多个单词或多个字符的便捷方法。下面是一些例子。

- 3dw：删除下面的 3 个单词。
- 5cl：修改下面的 5 个字母（即删除字母并进入输入模式）。
- 12j：向下移动 12 行。

在多数命令前加上数字只是重复执行这些命令。此时对于使用 vi 命令应该相当精通了。一旦习惯了使用 vi，就会发现其它文本编辑器使用起来效率都不高了。

在很多 Linux 系统中调用 vi 时，实际上正在调用 vim 文本编辑器，它运行在 vi 兼容 模式下。进行大量编程工作的人可能更愿意使用 vim，因为它以不同颜色显示不同的代码层 次。vim 还有一些其它有用的功能，例如在打开文档时，将光标放在最后一次退出文件时光 标所在的

位置。

vi 编辑器在开始时很难学，可是一旦掌握了它，就永远不必使用鼠标或功能键了—— 个键盘就可以快速高效的在文件中编辑和移动。

3.3.4 使用 GCC

1.GCC 简介

通常所说的 GCC 是 GUN Compiler Collection 的简称，除了编译程序之外，它还含其 他相关工具，所以它能把易于人类使用的高级语言编写的源代码构建成计算机能够直接执行 的二进制代码。GCC 是 Linux 平台下最常用的编译程序，它是 Linux 平台编译器的事实标准。同时，在 Linux 平台下的嵌入式开发领域，GCC 也是用得最普遍的一种编译器。GCC 之

所以被广泛采用，是因为它能支持各种不同的目标体系结构。例如，它既支持基于宿主的开发（简单讲就是要为某平台编译程序，就在该平台上编译），也支持交叉编译（即在 A 平台上编译的程序是供平台 B 使用的）。目前，GCC 支持的体系结构有四十余种，常见的有 X86 系列、Arm、 PowerPC 等。同时，GCC 还能运行在不同的操作系统上，如 Linux、Solaris、Windows 等。

除了上面讲的之外，GCC 除了支持 C 语言外，还支持多种其他语言，例如 C++、Ada、Java、Objective-C、FORTRAN、Pascal 等。

下面我们将介绍 Linux 平台下应用程序的编译过程，以及使用 GCC 编译应用程序的具 体用法，同时详细说明了 GCC 的常用选项、模式和警告选项。

2.使用 GCC 编译程序的过程

对于 GNU 通用编译器来说，程序的编译要经历预处理、编译、汇编、连接四个阶段， 如下图所示：

从功能上分，预处理、编译、汇编是三个不同的阶段，但 GCC 的实际操作上，它可以 把这三个步骤合并为一个步骤来执行。下面我们以 C 语言为例来谈一下不同阶段的输入和 输出情况。

在预处理阶段，输入的是 C 语言的源文件，通常为*.c*。它们通常带有*.h* 之类头文件的 包含文件。这个阶段主要处理源文件中的 *#ifdef*、 *#include* 和*#define* 命令。该阶段会生 成一个中间文件.i，但实际工作中通常不用专门生成这种文件，因为基本上用不到；若非 要生成这种文件不可，可以利用下面的示例命令：

```
GCC -E test.c -o test.i
```

在编译阶段，输入的是中间文件*.i*，编译后生成汇编语言文件.s 。这个阶段对应的GCC 命令如下所示：

```
GCC -S test.i -o test.s
```

在汇编阶段，将输入的汇编文件.s 转换成机器语言.o。这个阶段对应的 GCC 命令如下所示：

```
GCC -c test.s -o test.o
```

最后，在连接阶段将输入的机器代码文件 *.s（与其它的机器代码文件和库文件）汇集成一个可执行的二进制代码文件。这一步骤，可以利用下面的示例命令完成：

```
GCC test.o -o test
```

上面介绍了 GCC 编译过程的四个阶段以及相应的命令。下面我们进一步介绍常用 GCC的模式。

3.GCC 常用模式

这里介绍 GCC 追常用的两种模式：编译模式和编译连接模式。下面以一个例子来说明

各种模式的使用方法。为简单起见，假设我们全部的源代码都在一个文件 test.c 中，要想 把这个源文件直接编译成可执行程序，可以使用以下命令：

```
$ GCC -o test
```

这里 test.c 是源文件，生成的可执行代码存放在一个名为 test 的文件中（该文件是

机器代码并且可执行）。-o 是生成可执行文件的输出选项。如果我们只想让源文件生成目 标文件（给文件虽然也是机器代码但不可执行），可以使用标记 -c ，详细命令如下所示：

```
$ GCC -c test.c
```

默认情况下，生成的目标文件被命名为 test.o，但我们也可以为输出文件指定名称，

如下所示：

```
$ GCC -c test.c -o
```

上面这条命令将编译后的目标文件命名为 mytest.o，而不是默认的 test.o。

迄今为止，我们谈论的程序仅涉及到一个源文件；现实中，一个程序的源代码通常包 含在多个源文件之中，这该怎么办？没关系，即使这样，用 GCC 处理起来也并不复杂，见 下例：

```
$ GCC -o test first.c second.c third.c
```

该命令将同时编译 3 个源文件，即 first.c、second.c 和 third.c，然后将它们连接成一个可执行程序，名为 test。

许多情况下，头文件和源文件会单独存放在不同的目录中。例如，假设存放源文件的 子目录名为./src，而包含文件则放在层次的其他目录下，如 ./inc。当我们在./src 目录下 进行编译工作时，如何告诉 GCC 到哪里找头文件呢？方法如下所示：

```
$ gcc test.c –I../inc -o test
```

上面的命令告诉 GCC 包含文件存放在./inc 目录下，在当前目录的上一级。如果在编译时需要的包含文件存放在多个目录下，可以使用多个 -I 来指定各个目录：

```
$ gcc test.c –I../inc –I../../inc2 -o test
```

这里指出了另一个包含子目录 inc2，较之前目录它还要在再上两级才能找到。

另外，我们还可以在编译命令行中定义符号常量。为此，我们可以简单的在命令行中 使用-D 选项即可，如下例所示：

```
$ gcc -DTEST_CONFIGURATION test.c -o test
```

上面的命令与在源文件中加入下列命令是等效的：

```
#define TEST_CONFIGURATION
```

在编译命令行中定义符号常量的好处是，不必修改源文件就能改变由符号常量控制的行为。

实际上，GCC 命令提供了非常多的命令选项，但并不是所有都要熟悉，初学时掌握几个常用的就可以了，到后面再慢慢学习其它选项，免得因选项太多而打击了学习的信心。

为了方便读者查阅，这里将常见的编译选项列举如下：

(1) 常用编译命令选项 假设源程序文件名为 test.c。

I. 无选项编译链接 用法：#gcc test.c

作用：将 test.c 预处理、汇编、编译并链接形成可执行文件。这里未指定输出文件， 默认输出为 a.out。

II. 选项 -o

用法：#gcc test.c -o test

作用：将 test.c 预处理、汇编、编译并链接形成可执行文件 test。-o 选项用来指定 输出文件的文件名。

III. 选项 -E

用法：#gcc -E test.c -o test.i 作用：将 test.c 预处理输出 test.i 文件。

IV. 选项 -S

用法：#gcc -S test.i

作用：将预处理输出文件 test.i 汇编成 test.s 文件。

V. 选项 -c

用法：#gcc -c test.s

作用：将汇编输出文件 test.s 编译输出 test.o 文件。

VI. 无选项链接

用法：#gcc test.o -o test

作用：将编译输出文件 test.o 链接成最终可执行文件 test。

VII. 选项-O

用法：#gcc -O1 test.c -o test

作用：使用编译优化级别 1 编译程序。级别为 1~3，级别越大优化效果越好，但编译时 间越长。

(2) 多源文件的编译方法

如果有多个源文件，基本上有两种编译方法 （这里假设有两个源文件为 test.c 和 testfun.c。）。

I. 多个文件一起编译

用法：#gcc testfun.c test.c -o test

作用：将 testfun.c 和 test.c 分别编译后链接成 test 可执行文件。

II. 分别编译各个源文件，之后对编译后输出的目标文件链接。 用法：

```
#gcc -c testfun.c //将 testfun.c 编译成 testfun.o
#gcc -c test.c //将 test.c 编译成 test.o
#gcc -o testfun.o test.o -o test //将 testfun.o 和 test.o 链接成 test
```

以上两种方法相比较，第 1 种方法编译时需要所有文件重新编译，而第 2 种方法可以只重新编译修改的文件，未修改的文件不用重新编译。

4.警告功能

当 GCC 在编译过程中检查出错误的话，它就会中止编译；但检测到警告时却能继续编 译生成可执行程序，因为警告只是针对程序结构的诊断信息，它不能说明程序一定有错误， 而是存在风险，或者可能存在错误。虽然 GCC 提供了非常丰富的警告，但前提是你已经启 用了它们，否则它不会报告这些检测到的警告。

在众多的警告选项之中，最常用的就是 -Wall 选项。该选项能发现程序中一系列的常见 错误警告，该选项用法举例如下：

```
$ gcc -Wall test.c -o test
```

该选项相当于同时使用了下列所有的选项：

- unused-function：遇到仅声明过但尚未定义的静态函数时发出警告。
- unused-label：遇到声明过但不使用的标号的警告。
- unused-parameter：从未用过的函数参数的警告。
- unused-variable：在本地声明但从未用过的变量的警告。
- unused-value：仅计算但从未用过的值得警告。
- Format：检查对 printf 和 scanf 等函数的调用，确认各个参数类型和格式串中的 一致。
- implicit-int：警告没有规定类型的声明。
- implicit-function-：在函数在未经声明就使用时给予警告。
- char-subscripts：警告把 char 类型作为数组下标。这是常见错误，程序员经常忘记在某些机器上 char 有符号。
- missing-braces：聚合初始化两边缺少大括号。
- Parentheses：在某些情况下如果忽略了括号，编译器就发出警告。
- return-type：如果函数定义了返回类型，而默认类型是 int 型，编译器就发出警 告。同时警告那些不带返回值的 return 语句，如果他们所属的函数并非 void 类型。
- sequence-point：出现可疑的代码元素时，发出报警。
- Switch：如果某条 switch 语句的参数属于枚举类型，但是没有对应的 case 语句使 用枚举元素，编译器就发出警告（在 switch 语句中使用 default 分支能够防止这个警 告）。超出枚举范围的 case 语句同样会导致这个警告。
- strict-aliasing：对变量别名进行最严格的检查。
- unknown-pragmas：使用了不允许的#pragma。
- Uninitialized：在初始化之前就使用自动变量。

需要注意的是，各警告选项既然能使之生效，当然也能使之关闭。比如假设我们想要使用-Wall 来启用个选项，同时又要关闭 unused 警告，可以通过下面的命令来达到目的：

```
$ gcc -Wall -Wno-unused test.c -o test
```

下面是使用-Wall 选项的时候没有生效的一些警告项：

- cast-align：一旦某个指针类型强制转换时，会导致目标所需的地址对齐边界扩 展，编译

器就发出警告。例如，某些机器上只能在 2 或 4 字节边界上访问整数，如果在这 种机型上把 char 强制转换成 *int* 类型， 编译器就发出警告。

- sign-compare：将有符号类型和无符号类型数据进行比较时发出警告。
- missing-prototypes ：如果没有预先声明函数原形就定义了全局函数，编译器就 发出警告。即使函数定义自身提供了函数原形也会产生这个警告。这样做的目的是检查没有 在头文件中声明的全局函数。
- Packed：当结构体带有 packed 属性但实际并没有出现紧缩式给出警告。
- Padded：如果结构体通过充填进行对齐则给出警告。
- unreachable-code：如果发现从未执行的代码时给出警告。
- Inline：如果某函数不能内嵌（inline），无论是声明为 inline 或者是指定了- finline-functions 选项，编译器都将发出警告。
- disabled-optimization：当需要太长时间或过多资源而导致不能完成某项优化时 给出警告。

上面是使用-Wall 选项时没有生效，但又比较常用的一些警告选项。本文中要介绍的最后一个常用警告选项是-Werror。使用该选项后，GCC 发现可疑之处时不会简单的发出警告就算完事，而是将警告作为一个错误而中断编译过程。该选项在希望得到高质量代码时非常 有用。

3.3.5 使用 GDB

1.概述

GDB 是 GNU 开源组织发布的在 Linux 系统下用来调试 C 和 C＋＋程序的强力调试器，它 可以在程序运行时用来观察程序的内部结构和内存等的使用情况。

一般来说，GDB 主要帮助程序员完成以下 4 个方面的工作：

(1) 启动程序，可以按照自定义的要求运行程序；

(2) 在被调试的程序所指定的断点处停止；

(3) 程序停止时检查此时程序中所发生的事件；

(4) 动态的改变程序执行环境。

在命令行上输入“gdb”并按回车键就可以运行 GDB 了，如果一切正常的话 GDB 将被启 动并且将在屏幕上显示如下内容：

```
#gdb
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

当启动 GDB 后，在命令行上可以指定很多的选项，也可以下面的形式来运行 GDB：

```
GDB &lt;程序名&gt;
```

当用这种方式运行 GDB 时，能直接指定想要调试的程序，GDB 将装入某个可执行文件。

GDB 也可以检查一个因程序异常终止而产生的 core 文件。

为了使 GDB 正常工作，必须使程序在编译时包含调试信息。调试信息包含程序里的每 个变量的类型和在可执行文件里面的地址映射以及源代码的行号， GDB 利用这些信息使源代 码和机器码相关联。在编译用“-g”选项打开调试选项。

2.GDB 基本命令

GDB 支持很多的命令，能实现不同的功能。表 2-1 列出了 GDB 调试时常用的一些命令。

表 3-8 GDB 基本命令

命令	含义
backtrace	显示函数调用的所有栈框架的踪迹和当前函数的参数值
break	设置一个断点，这个命令需要指定代码行或者函数名作为参数
clear	删除一个断点，这个命令需要指定代码行或者函数名作为参数
continue	在调试器停止的地方继续执行
Ctrl+C	在当前位置停止执行正在执行的程序，断点在当前行
disable	禁止断点功能，这个命令需要禁止的断点在断点列表索引值作为参数
display	在断点停止的地方显式指定的表达式的值
enable	允许断点功能，这个命令需要禁止的断点在断点列表索引值作为参数
finish	继续执行，直到当前函数返回
ignore	忽略某个断点制定的次数。例如：“ignore4 23”忽略断点 4 的 23 次运行，在第 24 此运行时中断
Info breakpoints	查看断点信息
Info display	查看设置的需要显式的表达式的信息
kill	终止当前 debug 进程
list	显示 10 行代码。如果没有提供参数给这个命令，则从当前行开始显示 10 行代码。如果提供了函数名作为参数，则从函数开头显示。如果提供代码行的编号作为参数，这一行作为开头显示
load	动态载入一个可执行文件到调试器
next	执行下一行的源代码的所有指令。如果是函数调用，则也当作一行源代码，执行到此函数返回
nexti	执行下一行源代码中的一条汇编指令
print	显式变量的值
ptype	显示变量的类型

return	强制从当前函数返回
run	从程序开始的地方执行
rwatch	指定一个变量，如果这个变量被读，则暂停程序运行，在调试器中显示信息，并等待下一个调试指令，参考 rwatch 和 watch 命令
set	设置变量的值。例如："set nval=54"将把 54 保存到 nval 变量中
step	继续执行程序下一行源代码的所有指令。如果是调用函数，这个命令将进入函数的内部，单步执行函数中的代码
stepi	继续执行程序下一行源代码的汇编指令。如果是调用函数，这个命令将进入函数的内部，单步执行函数中的代码
txbreak	在当前函数的退出点上设置一个临时断点（只可使用一次）
undisplay	删除一个 display 设置的变量显示，这个命令需要将 display list 中的索引作为参数
watch	指定一个变量，如果这个变量被写，则暂停程序运行，在调试器中显示信息，并等待下一个调试命令，参考 rwatch 和 watch 命令
whatis	显示变量的值和类型
xbreak	在当前函数的退出点上设置一个断点
awatch	指定一个变量，如果这个变量被读写，则暂停程序运行，在调试器中显示信息，并等待下一个调试命令，参考 rwatch 和 watch 命令

3.GDB 的操作

GDB 支持很多与 shell 程序一样的命令编辑特征。能像在 Bash 里那样按"Tab"键让 GDB 补齐一个唯一的命令。如果该命令不唯一的话， GDB 会列出所有匹配的命令，也能用光 标键上下翻动历史命令。更为详尽的内容，请查阅 GDB 的帮助。

4.GDB 应用举例

本小节通过一个实例一步步的用 GDB 调试程序。下面是将被调试的程序，这个程序被 称为 greeting.c，它显示一个简单的问候，再用反序将它列出。

```
#include &lt;stdio.h&gt;
#filename.greeting.c

main()
{
    char my_string[] = "hello there";
    my_print (my_string);
    my_print2 (my_string);
}

void my_print(char *string)
{
    printf("The string is %s\n",string);
}

void my_print2(char *string)
{
    char *string2;
    int size,i;
    size = strlen(string);
    string2 = (char *) malloc(size+1);
    for ( i=0;i&lt;size;i++)
    {
        string2[size -i] = string[i];
        string2[size+1] = '\0';
        printf("The string printed backward is %s\n",string2);
    }
}
```

用命令编译它：

```
#gcc -o test test.c
```

这个程序执行时显示如下结果：

```
The string is hello there
The string printed backward is
```

输出的第一行是正确的，但第二行打印出来的东西并不是所期望的，所设想的输出应该是：

```
The string printed backward is ereht olleh
```

由于某些原因，my_print2 函数没有正常工作。用 GDB 查看问题究竟出在哪儿，先输入如下命令：

```
#gdb greeting
```

注意，在编译 greeting 程序时需要把调试程序打开

如果在输入命令时忘了把要调试的程序作为参数传递给 GDB，可以在 GDB 提示符下用 file 命令载入它：

```
(gdb) file greeting
```

这个命令将载入 greeting 可执行文件，就像在 GDB 命令行里装入它一样。

这时可以用 GDB 的 run 命令运行 greeting 了，其在 GDB 里被运行后结果如下：

```
(gdb) run
Starting program: /root/geeeting
The string is hello there
The string printed backward is
Program exited with code p41
```

这个输出和在 GDB 外面运行的结果一样，但为什么反序打印却没有工作呢？为了找出症结所在，可以在 my_pringt2 函数的 for 语句后设一个断点，具体的做法是在 gdb 提示符 下键入 list 命令 3 次，列出源代码，如下所示：

```
(gdb) list
(gdb) list
(gdb) list
```

第一次输入 list 命令的输出如下：

```
#include &lt;stdio.h&gt;
main()
{
    char my_string[] = "hello there";
    my_print(my_string);
    my_print2(my_string);
}
```

如果按下回车，GDB 将再执行一次 list 命令，输出如下：

```
my_print(char *string)
{
    printf("The string is %s\n",string);
}
my_print2(char *string)
{
    char *string2;
    Int size,i;
}
```

再按一次一次回车将列出 greeting 程序的剩余部分，如下所示：

```
size = strlen(strint);
string2 = (char*)malloc(size+1);
for(i=0;i&lt;size;i++)
{
    string2[size-i] = string[i];
    string2[size+1] = '\0';
    printf("The string printed backward is %s\n",string2);
}
```

从列出的源程序可以看到要设断点的地方在第 24 行，在 GDB 命令行提示符下输入如下命令设置断点：

```
(gdb) break 24
```

GDB 将作出如下的响应：

```
Breakpoint 1 at 0x139: file greeting.c,line24
(gdb)
```

现在再输入 run 命令，将产生如下的输出：

```
Starting program:/root/greeting
The string is hello there
Breakpoint 1,my_print2(string = 0xbfffdc4 "hello there") at greeting.c :24
24 string2[size-i] = string[i];
```

能通过设置一个观察 string2[size-i]变量值的观察点来查看错误是怎样产生的，输入一下指令：

```
(gdb) watch string2[size-i];
```

GDB 将作出如下回应：

```
Watchpoint 2:string2[size-i]
```

现在可以用 next 命令一步一步的执行 for 循环了，如下所示：

```
(gdb) next
```

经过第一次循环后：GDB 告诉我们 string2[size-i]的值是 h。GDB 用如下的显示来告诉这个信息：

```
Watchpoint 2,string2[size-i]
Old value=0 '\000'
New value = 104 'h'
my_print2(string = 0xbfffdc4 "hello there") at greeting.c:23
23 for ( i= 0;i&lt;size; i++ )
```

这个值正是期望的，后来的数次循环的结果都是正确的。当 i=10 时，表达式string2[size-i]的值等于 e，size-i 的值等于 1，最后 1 个字符已经复制到字符串里了。

如果再把循环执行下去，会看到已经没有值分配给 string2[0]了，而它是字符串的第 1 个字符，因为 malloc()函数在分配内存时把它们初始化为空 (null)字符，所以 string2 的第 1 个字符是空字符，这就解释了打印 string2 时没有任何输出的原因。

现在找出了问题在哪里，改正这个错误很容易。需要把代码里写入 string2 的第 1 个 字符的偏移量改为 size-1。

使代码正常工作有很多种修改方法。一种是另设一个比实际大小小 1 的变量，这种解 决办法的代码如下：

```
#include &lt;stdio.h&gt;

main()
{
    char my_string[] = "hello there";
    my_print(my_string);
    my_print2(my_string);
}

my_print(char *string)
{
    printf("The string is %s\n",string);
}

my_print2(char *string)
{
    char *string2;
    int size,size2,i;
    size = strlen(string);
    size2 = size-1;
    string2 = (char*)malloc(size+1);
    for( i = 0;i&lt;size;i++ )
    {
        string2[size-i] = string[i];
    }
    string2[size] = '\0';
    printf("The string printed backward is %s\n",string2);
}
```

3.4 Mac 编程基础

这里我们重点讲述在 Mac OS X 上编程相关的技能。

3.4.1 你必须掌握的技能

一些周知的行业基础技能就不多说了，像标准 C++、计算机英语这些都是必需的，下面 是在 Mac 上做开发需要的技能：

- 熟练使用 Mac OS X 系统
- 了解 Mac OS X 的系统架构
- 熟悉 Mac OS X 文件系统
- 了解 Carbon 环境
- 熟悉 Cocoa 环境
- 熟练使用 Xcode
- 能够使用其它的编程工具辅助开发

3.4.2 Mac OS X 的系统架构

首先我们通过下面的图 3-6 了解一下 Mac OS X 系统的主要架构，它忽略了一些细 节，但是比较清晰的描绘了系统的主要组件和相互间的关系。

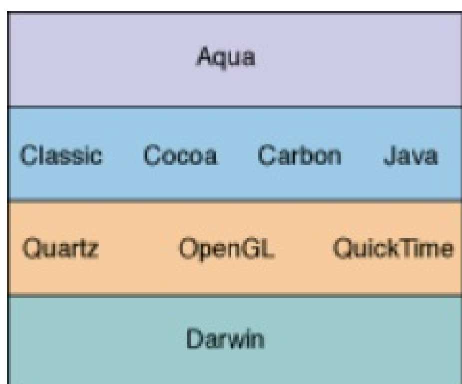


图 3-6 Mac OS X 系统架构简图

同现代其它的操作系统一样，在其内部结构上， Mac OS X 的是积木式的层次集合。这 种结构的特点是，系统的较低层实现包含有上层软件所依赖的基础服务，比如 OpenGL 框架 可能担负底层绘图任务，而内核层则负责任务调度或硬件管理。

图 3-6 给出的是 Mac OS X 的系统架构图，从中可以看到，系统最底层主要包含 Darwin，它是系统的核心；它的上层是绘图与媒体层，为播放音频视频、渲染 2D/3D 图形 提供了强大而专门的服务；再往上则是系统的 Framework 层，包含 Cocoa、Carbon 等应用 程序框架，这

里是我们要学习的主要内容；最上层则是用户层体验层，包含 Aqua 和 Spotlight 等)令 Mac OS X 与众不同的方法、技术和应用程序， 其中 Aqua 是 Mac OS X 的用户界面， 对于一般的使用用户而言， Mac OS X = Aqua。

更为详细的 Mac OS X 分层结构请参见图 3-7。

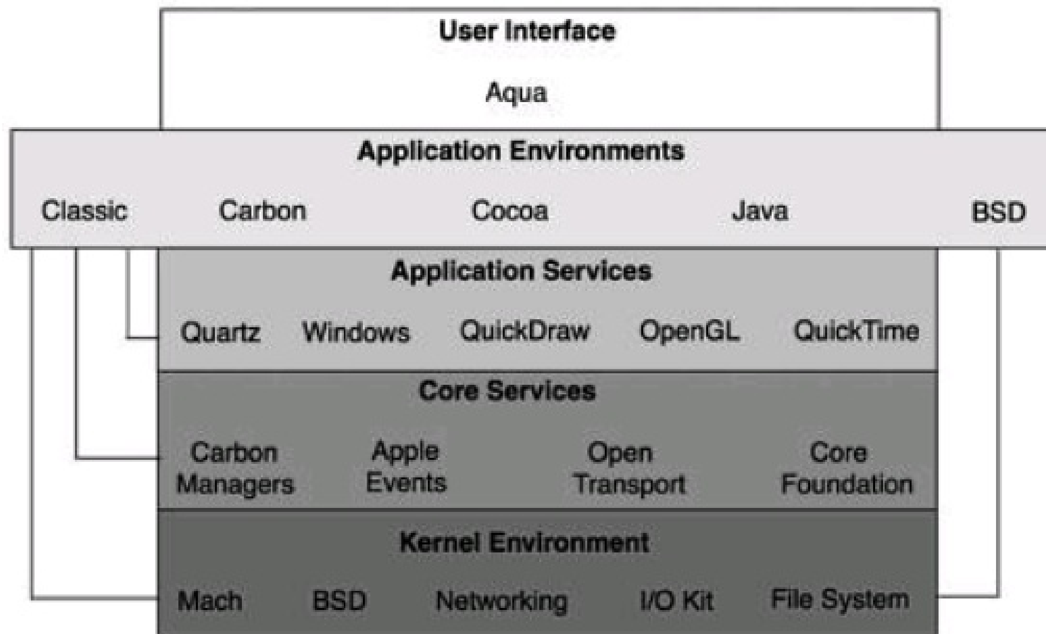


图 3-7 Mac OS X 架构图

3.4.3 Mac OS X 文件系统

在任何一个操作系统当中，文件系统都是非常重要的部分，毕竟每个用户都需要在文 件系统中保存自己的资料。在 Mac OS X 系统中，文件系统的组织结构扮演了非常重要的角 色，它能够帮助用户找到想要的文件。这种组织结构同时也让应用和系统本身在为满足用户 需要而查找文件资源的时候更加简单快捷。

Mac OS X 中的文件系统内核中有一套从 BSD 操作系统中继承而来的目录结构体系。 虽然大部分目录都被 Finder 所隐藏,但是 BSD 世界中的很多元素都显露无遗。其中文件权 限模型、符号链接以及用户目录等概念都来自于 BSD。Mac OS X 系统还增加了一些它自己 的概念用来向用户提供一个安全精致的环境对文件和文件夹进行管理。

Mac OS X 在满足了用户期望的易用性的同时，它的设计目标也提供了强大和灵活的功 能特点。在这点上，这种文件系统向用户提供一种统一的目录结构，从而使用户可以很容易 的找到资源。(这种统一风格对开发者也非常有用，他们开发的应用有时需要知道去哪里能 找到某些重要的资源。) 还有其它的一些文件系统固有风格，比如别名、扩展名隐藏和显示 名称等等都能够改善用户的使用感受。

Mac 文件系统涉及的内容很多，这里只介绍与 Qt 开发有紧密联系的部分。 1.文件系统域 在多用户系统中，控制用户对系统资源的访问对于维护系统的稳定是非常重要的。 Mac OS X 系统定义了好几种文件系统域，每种文件系统域都提供了固定的文件夹用于存储系统资源。

在每种域当中，对资源的访问权限是由当前用户的权限决定的。

一共有四种文件系统域，用户域、本地域、网络域和系统域。

(1) 用户域

用户域中包含了某个用户专有的资源。用户域由当前登录用户的 home 目录表示。Mac OS X 计算机的每个用户在这个计算机上或者这个计算机所链接的本地局域网中都应该有一 个帐号。每个用户帐号在文件系统中都有一个分配的空间，这个空间被叫做用户 home 目 录。这个目录中存放了用户的程序、资源以及文档。用户 home 目录的名称基于用户的登录 名称，该名称必须是唯一的。

用户域使每个用户拥有一个可以定制的工作环境成为可能。在用户登录的时候， Finder 会根据用户域中的偏好设置，载入该用户上次的工作环境和设置。类似的，应用程 序以及其它系统软件也能够利用用户域中的信息载入：应用偏好设置、网络设置、电子邮件 设置、字体设置、ColorSync 配置以及其它设置。

用户 home 目录的位置取决于用户帐号。如果用户帐号是计算机的本地帐号，那么用 户的 home 目录就在启动卷的 Users 目录当中。而如果用户帐号是一个网络帐号，那么 home 目录就在网络服务器上。如果不考虑 home 目录的物理位置，Mac OS X 还沿用了 UNIX 系统的惯例，在某些时候用~ (波浪号)字符来表示用户的 home 目录。这个波浪号可 以和其它目录名或者用户名结合起来使用，表示特定的用户目录。表 3-9 对这个概念有所 说明。

表 3-9 利用波浪号表示 home 目录中的位置

~	当前用户 **home** 目录的最高一层
~/Library/Fonts	当前用户 home 目录中保存字体的目录
~Steve	用户 Steve 的 home 目录最高一层

每个新用户的 home 目录都会包含一些缺省目录和资源。 表 3-10 列出了你可能会在 用户 home 目录中看到的一些常见目录。

表 3-10 Home 目录的内容

用户目录	描述
Applications	包含仅仅当前用户可用的应用
Desktop	包含了 Finder 在当前登录用户桌面上显示的桌面项
Documents	包含了用户的个人文档
Library	包含了应用设置、偏好设置一起其他用户专有的系统资源
Movies	包含了 QuickTime 以及其它格式的数字影片
Music	包含数字音乐文件 (.aiff、.mp3、.m4p 及其它格式)
Pictures	包含各种格式的图像文件
Public	包含了用户需要和其他用户共享的内容。缺省情况下，其他用户可以访问这个目录
Sites	包含了用户个人网站的网页。如果需要其他用户能够访问这些网页，需要使 Web 共享

当用户帐号创建时，Applications 目录不会自动创建到他的 home 目录中。不过，用户可以自己创建一个 Applications 目录，并把自己的应用放进去。系统会自动在这个位置 搜索应用。

系统通过一系列缺省权限保护用户 home 目录中的文件和目录不受外界影响，用户也 可以在任何时候改变这个缺省权限设置。用户创建的任何目录也都会继承其上层目录的权限 设置。

除了每个用户的 home 目录，Users 目录也包含一个 Shared 子目录。本地计算机上 的所有用户都可以访问该目录，只有用户可以访问该目录，应用不能将自己的内容存放在这 里，除非用户指定这么作。任何用户都可以往这个目录中写入文件、从中获取文件或者读取 其中的文件。尽管这个目录与用户域无关，但是它为用户之间交换文档和其它文件提供了一 个方便的途径。

(2) 本地域

本地域中包含了本地可用但是系统运行时不需要的那些资源。本地域中的资源通常会包含应用、工具、定制字体、定制启动项以及全局应用设置。本地域并不会对应一个单独的物理目录，而是包含了本地 boot (和 root)卷中的几个目录。拥有管理员权限的用户可以 在这个域中增加、删除和修改内容。 root 卷中的 Applications 和 Library 目录包含了本地域的资源。计算机系统的当前用户是可以访问这些资源的，但是网络中的其它计算机上的 用户无法访问。

计算机管理员如果需要这个系统中的所有用户共享一些资源的话，他可以把这些资源 安装到本地域当中。Apple 系统中应用都在/Applications 和 /Applications/Utilities 目录中。第三方应用和工具也应该在这个目录中。其它的系统资源，比如字体、 ColorSync 配置、偏好设置以及插件都应该安装在 Library 目录下适当的子目录中。

(3) 网络域

网络域中的资源包括：本地局域网中的所有用户共享的应用和文档。这个域中的内容通常都位于网络文件服务器，并且在网络管理员的控制之下。 表 3-11 显示了网络域中的部分目录。

表 3-11 网络目录

位置	描述
/Network/Applications	包含了本地网络中所有用户都能够运行的应用
/Network/Library	包含了一些本地网络中所有用户都可以使用的资源，如插件、文档、框架、色彩以及字体
/Network/Servers	包含了组成本地网络的所有 NFS 文件服务器的挂接点
/Network/Users/	包含了所有本地网络用户的 home 目录。这是 home 目录的缺省路径

(4) 系统域

系统域包含了 Mac OS X 相同运行时候需要系统软件和资源。系统域中的所有资源都 位于 root 卷的 /System 目录中。这些资源都由 Apple 提供，只有 root 用户可以修改该 目录中的内容。

每个资源所属的域说明了系统用户对这个资源是否可以应用或者访问。比如，一个安 装在某用户 home 目录当中的字体只能被该用户使用。如果一个管理员在网络域中安装相同 的字体，那么所有网络用户就都可以访问它了。

缺省情况下，/System 目录只包含了一个 Library 子目录。这个子目录中包含了许多 与系统中其它库目录中相同类型的资源。不过，在系统域中，这个目录中还包含了一些核心 服务、框架以及组成 Mac OS X 系统的应用。

Mac OS X 系统对于每个域都提供了一套初始目录用来组织其中包含的资源。 Mac OS X 系统在不同的域之间都会使用相同的目录名称保存相同类型的资源。这种名称一致性能够使 用到这些资源的用户或者系统更加容易的查找资源。当系统需要查找某个资源的时候，它会 依次查找各个域，直到找到想要的资源。搜索开始与用户域，然后依次查找本地域、网络域 和系统域。

2.库目录

库目录是一个专门用来存储与应用相关的和与系统相关的资源的目录。每个文件系统域都有一个它自己的库目录拷贝，其访问级别与域类型是相匹配的。尽管应用可以利用这个目录存储内部数据或者临时文件，可是该目录并不是用来存储应用包本身以及用户数据文件 的。应用包存在于某个适当的/Applications 目录，而用户数据存在于用户的根目录下。

库目录还包含了许多标准的子目录。由于系统例程会假定其中许多子目录是存在的， 所以尽量不要删除这些标准目录。

表 3-12 列举了对开发人员来说最相关的目录。你可以根据这个表来确定在哪里存储你 的软件所需要的文件，当然这个表并不是完整的。

表 3-12 库目录的子目录

Application Support	包含了应用相关的数据以及支持文件，比如第三方的插件，帮助应用，模板以及应 用使用到但是并不需要用来支持运行的额外资源文件。按照惯例，所有这些内容都 会被存储在以应用名称命名的子目录当中。比如，**MyApp** 应用所用到的第三方资源 应该在 **Application Support/MyApp/** 目录中。需要注意，必须的资源文件应该 在应用包当中
Assistants	包含了帮助用户进行配置或者其它任务的程序
Components	包含了系统包和扩展
Contextual Menu Items	包含了用于扩展系统级菜单的插件
Documentation	包含了供计算机用户和管理员参考的文档文件和 Apple 帮助包。(Apple 帮助包在 Help 子目录当中。) 在本地域中，这个目录包含了 Apple 公司发布的帮助包(不 包括开发者文档)。
Keyboards	包含了键盘定义

3.开发者目录

Xcode 工具 CD 中包含了开发 Mac OS X 系统软件所需要的应用、工具、文档以及其 它资源。开发者可以在安装 Mac OS X 系统的时候分别安装以上工具。在安装这些工具的时 候，安装程序会把所有的软件组件放在启动卷的 /Developer 目录中。表 3-13 列举了这个 目录的内容。

表 3-13 发者目录的子目录

目录	内容
Applications	包含了用来管理和构建软项目的应用。这些工具包括 Xcode 以及接口构造器，它们可以用来窗代码和生 成接口。它还包含了一套性能工具、Java 工具、图形工具以及通用工具
Documentation	包含了另外一些与开发者相关的文档
Examples	包含了按照通用类型组织起来的范例项目。这些项目都是完全可以工作的，这些可以编译运行的项目可以 帮助你更多的了解 Mac OS X 系统
Makefiles	包含了用来编译和转换遗留项目的 makefiles 以及 jamfiles
SDKs	包含了用来为旧版本 Mac OS X 系统开发软件的软件开发工具包。每个 SDK 都包括 Mac OS X 特定版本 的头文件和 stub 库
Tools	包含了命令行开发实用工具，包括那些用于创建和操作 HFS 资源 forks 的工具

4.文件操作命令

再次提醒，Mac OS X 是类 Unix 系统，所以它的绝大多数文件操作命令与 Linux 是一致 的，这里不详细展开，介绍几个实用的技巧。

- 在 Mac OS X 中是区别大小写字符的 比如 A.txt 不等于 a.txt。
- 系统的根目录标志 / 并不是可有可无的

  cd /System 表示转到根目录下的 System 中，而 cd System 表示转到当前目录下的 System 中，请大家注意其中的区别。

- 如何进入命令行操作模式

  在图形界面下，用 finder 打开 应用程序 》实用程序》终端 如果连图形界面都进不去了（比如安错了显示驱动），开机时按 F8，用 `-s` 参数启动，然后输入命令 `mount -uw /`

- 用 Tab 键自动补齐命令

  比如你想到 /System 目录中去，输入 cd /Sy 然后按一下 Tab 键，命令就会自动补齐成 cd /System

- 操作带名字中带有空格的文件和目录

  空格在命令中写成 \空格， 比如要进入 My Documents，命令为 cd My\ Documents

- 查看命令的详细帮助

  命令是：man 命令名，比如要看看 ls 命令的详细用法，执行 man ls。

需要特别指出的是，大家不要有 Windows 下的盘符概念。Mac OS X 的所有文件都挂在根目录 / 下面。硬盘都挂在 /Volumes 下。比如你的移动硬盘叫做 USBHD，那么接上后桌面上会显示出一个硬盘图标，它实际在哪里呢？你在终端里执行 ls /Volumes/USBHD, 看看 显示出的是不是这个移动硬盘的内容。表 3-14 列出了常见的一些目录：

表 3-14 Mac OS X 常见目录

根目录位置	/
驱动所在位置	/Systme/Library/Extensions
用户文件夹位置	/User/用户名
桌面的位置	User/用户名/Desktop
硬盘挂载位置	/volumes/YourHdName

3.4.4 Cocoa 应用开发简介

Cocoa Framework 简称 Cocoa，是 Mac OS X 上原生支持的应用程序开发框架 。从 1989 年至今， Cocoa 一直在努力改进和完善。它设计优美，适合 RAD(Rapid Application Development）。无论你是资深的 Mac 开发“老鸟”，还是即将踏入 Mac 开发世界的新人，Cocoa 都是构建 Mac OS X 应用程序最强大、最高效的工具之一。

Cocoa 应用程序已经逐步成为 Mac OS X 的规范。 例如 iPhoto 和 Safari 都 是 Cocoa 应用程序，这些应用程序以设计巧妙的聪明的设计，丰富的功能以及吸引人的用户界 面得到了广泛的好评，而这些归根结底都是 Cocoa 的功劳。

1.Cocoa 环境

与许多的应用程序环境一样， Cocoa 同时拥有一个运行时间外观和一个开发外观。在 它的运行时间外观里， Cocoa 应用程序呈现 Aqua 的外观，并且能够与操作系统的其它部 分如可视化部件、应用程序等紧密结合。

Cocoa 的开发外观是一套集成的面对对象的软件构件，它使得开发人员创建 Mac OS X 应用程序成为一件容易和快乐的事情。使用 Cocoa，你可以在 Mac OS X 的世界里“为所欲 为”。

在开发 Cocoa 软件时主要的语言是 Objective-C， Objective-C 是 ANSI C 的一个超集，它扩展了特定的语法和语意功能（从 Smalltalk 衍生出来）来支持面对对象的编程。添加的不多的约定都比较简单而且容易学习和使用。因为 Objective-C 以 ANSI C 为 基础，您可以自由的混合使用纯 C 语言代码和 Objective-C 代码。

2.Cocoa 在 Mac OS X 中的位置

图 3-8 准确的描述了 Cocoa 在 Mac OS X 架构中的位置。Mac OS X 由一系列软件层组 成，自下而上分别是最底层的 Darwin， 中间的“核心服务”和“应用服务”层，上层软 件依赖下层软件提供的服务才能运行。 Mac OS X 中全部组成部分，包括 Cocoa， 最终都依 赖 Darwin 才能工作。

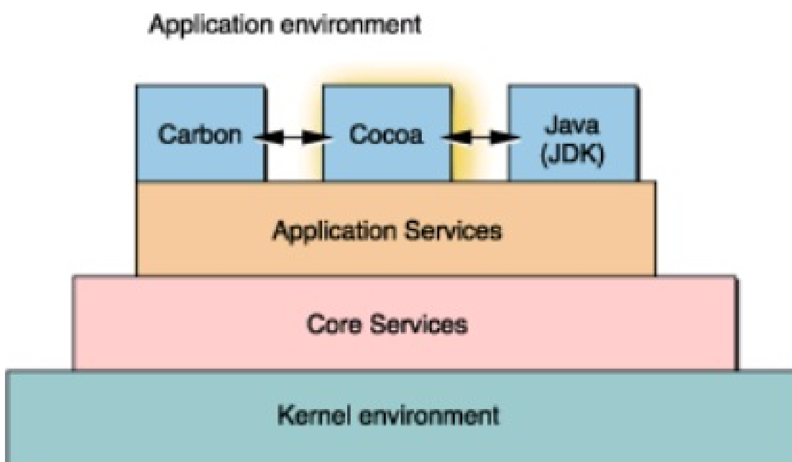


图 3-8 Cocoa 在 Mac OS X 的架构中

3.4.5 Xcode 简介

Xcode 是 Apple 自己开发的，只运行在 Mac OS X 平台下的 IDE。如果你想真正了解 Mac OS X 平台上的开发，就应该了解它。当然， Mac OS X 实现了 POSIX，固然也支持传统的 UNIX 编程环境；Eclipse 等 Java 开发工具也有 Mac OS X 版。不过这些，你在 Linux 或者

Windows 平台也能体验到。注意，Xcode 只是一个 IDE，Apple 并没有自己专属的编译器。事实上创建项目的时候，Xcode 仍然是在调用 GCC 命令。所以如果你很熟悉 UNIX 下的编程，你完全可以不用 Xcode，在命令行编译 Cocoa 应用程序，也是很有乐趣的。

Xcode 功能强大，它支持软件生命周期的全部过程，一旦使用，你就会很难放弃它。事 实上，目前 Xcode 几乎成为了多数 Mac 开发者的唯一选择。非常开心的是 Xcode 是免费 的，只要你有一台 Mac，随机带的安装盘里就 有 Xcode，升级也是免费的，不过如果你用的 是 Mac OS 10.5 以前版本的 OS，你将不能运行 Xcode 3.0 或以上的版本，而我们恰恰推荐 你使用 Xcode3.0，所以要很好的选择你的 Mac OS X 的版本。

Xcode 编译用 C, C++, Objective-C, Objective-C++, 和 Java 编写的源代码工程。 它产生所有 Mac OS X 上支持的可执行代码类型，包括命令行工具，框架，插件，内核扩 展，包，和应用程序。 Xcode 允许几乎无限制的自定义编译和调试工具，执行代码打包（包括信息属性列表和本地化的包），编译过程（包括拷贝文件，脚本文件和其它编译阶段），以及用户界面（包括分开的，多视图的代码编辑器）。如果您的代码在一个 CVS 库里， Xcode 提供了源代码控制，允许您添加文件到库里，确定修改，获得更新的版本，以 及比较版本。图 3-9 展示了一个在 Xcode 里的工程的例子。

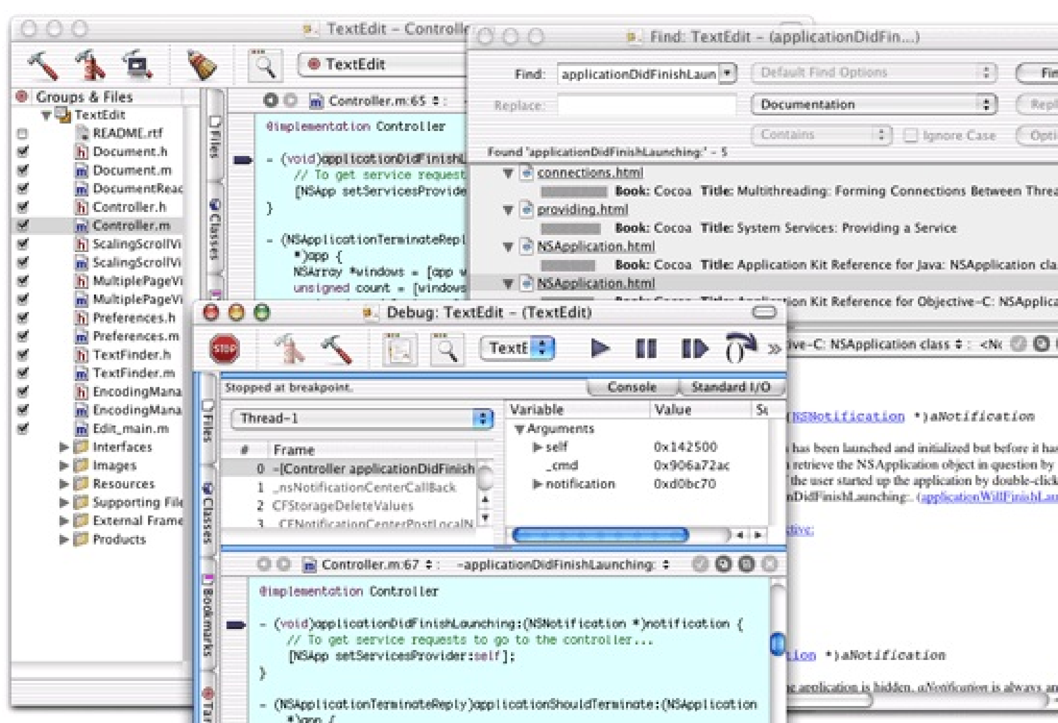


图 3-9 Xcode 中的 TextEdit 工程例子

Xcode 特别适合于 Cocoa 开发。在您创建一个工程的时候， Xcode 使用对应于 Cocoa 工程类型的工程模版为您设置一个起始的开发环境，工程类型有：应用程序（ Objective-C 或者 Java ），基于文档的应用程序（ Objective-C 或者 Java ），工 具，包，和框架。 Xcode 使用 GNU C 编译器（ gcc ）编译 Cocoa 软件，使用 GNU 源代码级别调试器（ gdb ）调试这些软件。 gcc 和 gdb 在 Cocoa 还是 NeXTSTEP （请参考 “历史简介”）的时候就被用于

Cocoa 的开发，经过若干年一直被精雕细琢，扩展，并且 调优以支持 Cocoa 二进制代码的编译和调试。 Xcode 还有一个类浏览的功能，您可以察看所有输入的 Cocoa 框架和任何您的自定义类，这些类按照它们继承关系排列；在类浏览器 里您可以请求任何类的文档。

要获得更多信息，请参考 Xcode 在苹果开发人员联盟网站（ http://www.apple.com.cn/developer/ ）上的 Xcode 主页，在那里您可以链接到 Xcode 最新的文档，这里就不再赘述了。

3.5 问题与解答

问：有了 MFC，还可以使用 Qt，我还有学习 Windows API 编程的必要吗？

答：应用 Windows API 编程时，要依据在前面讲解的基本步骤，而且要详细考虑到对 可能接收到的消息的具体处理。应用 Windows API 编程是可视化编程的基础，不像直接利

用 MFC 编程。利用 MFC 编程时，部分框架的代码是由该类库自动生成的， MFC 自己定义了一套编程模式，并且对自己实现的机制隐藏的比较深，用户必须遵守才不致出现问题；Qt 与 MFC 类似，它也是一套类库，用以帮助开发者快速构建应用，但是 Qt 使用标准 C++，并且做了扩展，它同样为使用者封装了很多底层的机制，但当你需要深入的开发一些功能时，你 仍然需要对本地平台有深入的了解；而利用 Windows API 函数进行编程，所有的代码需要用户自己完成，虽然编写程序比较繁杂，但能从根本上理解可视化编程的体系。所以，无论你是使用 MFC 还是 Qt 类库，都需要掌握 Windows API 编程。

问：Mac OS X 与 Linux 的异同

答：这个问题还是很有必要说一下，事实上很多使用 Mac OS X 的朋友也发现了，如果 你很熟悉 Linxu 下的命令行操作，在 Mac 的终端下也会一样的控制自如，其重要的原因之 一就是两个操作系统确实有密切的联系。

简单说，Mac OS X 是在 BSD 系统的基础上发展起来的，可以叫做 Darwin BSD，是基于 DarwinBSD 的类 UNIX 发行版，所以它的架构是类似 Linux 的。但是它并不开源， 借助于与 UNIX 的亲缘关系，它同样也可以使用大量的 GNU 的开源软件

linux 和 MacOS 是同一源头 ，都是属于类 UNIX 系统家族 。

问：我刚入手了 MAC，想在上面编程，可是听说 MAC 上的写程序就是相对很弱了,找了 很久也没找到 VC,VB 什么的,真的这么难吗？

答：不是这样的，我想你首先要了解的是， MAC OS X 是类 Unix 系统，所以在它上面编 程与在 Linux 上是类似的，但也有一些区别 ，所以 Unix/Linux 上的编程工具通常都能在 Mac 上使用。最开心的就是你同样可以使用 Qt。在 Mac 上你可以编写几乎所有的高级编程 语言程序, 如 C, C++, Cocoa, Java, Fortran, python, ruby,等等。喜欢手写代码的话， 你可以使用 gcc/g++编译器, ；喜欢用集成开发环境的话, 你可以用 Xcode + Interface Builder, 都是免费的。慢慢的，你会发现在 Mac 下的编程并不比 Windows 上弱，甚至某些 方面更方便、更强大。

另外要指出的是，Microsoft, Visual Studio 是 Windows 平台专属的 IDE，是商业授 权的产品，并且不能跨平台使用,你在 Mac 上是找不到它的。

问：Xcode 是不是只能在 Mac 上运行，Windows 上能不能用？要学编程的话是不是只用 学 C++就行了？

答：Xcode 只能在 Mac 上运行，但写出来的程序可以在 Mac 上运行，也能运行在 iPhone 上。

Xcode 支持多种语言，例如 C/C++/Java/AppleScript Studio 等，你还可以去另装其他 语言的支持，如 Pascal。但是用 Xcode 编程，语言方面的最佳选择是 Objective-C，它是 苹果 Cocoa 架构应用程序的所谓“母语”。

学习什么语言并不是十分重要，关键是要掌握编程的基本原理，熟练使用一种语言如 C++后，有时间再学习其它的语言，就会很快的触类旁通了。

问：请推荐几个比较好的讨论 Mac 编程的中文论坛？

答：关于 Mac 的综合性的论坛还是不少的，但大多数很少涉及编程开发的话题， 这其 中的中文论坛更是少见。以下论坛还不错，在线人数很多，但是繁体中文的， 供你参考。

- http://www.sinomac.com/
- http://www.imacguru.com

问：在 Mac 上编程的话，我是要侧重掌握 Carbon 还是 Cocoa 呢？

答：Carbon 主要是为 os 9 到 os x 过渡的应用所采用的 C 和 C++环境，并非面向对 象。carbon 应用能在 os 9 和 x 都能运行。但随着 os x 的出现，使用 carbon 编程的人正在 逐渐减少。

Cocoa 是比较先进的 OOP 环境，可以写较少的程序编出较复杂的应用。应用语言主要是 Objective-C 或标准 Java,也可以使用其它语言。它也是一种事件驱动的应用平台，是特别 为 Mac OS X 设计的一套面向对象的 Framework，因此无法在 Mac OS 8/9 上面运行。

但是，Carbon API 所写的老程序移植到 Mac OS X 会相对容易，而仅仅在部分情况 下，Mac OS X 下的使用 Carbon API 的程序可以不加修改在 Mac OS 8/9 上面运行

所以，我推荐你重点掌握 Cocoa，但是了解和学习一下 Carbon 上的知识与应用并没有 害处。

3.6 总结与提高

这一章不好写，笔者前后用了 1 个月时间才写成。但这些内容都是在 Qt 编程过程中必 需的，如果没有这些做基础，你的技能链是不连续的，做起开发来经常会遇到相关的问题， 导致效率不高。

在这一章里面，首先为大家介绍了一些编程的基础知识，然后再讲解一些与平台相关 的技能，主要包括 Windows、Linux 和 Mac OS X 平台，它们是目前 Qt 编程中可能会接触的 最主要的平台。由于本书主要讲解的是桌面平台的开发，所以对嵌入式平台没有作过多的涉及，有需求的朋友请参阅帮助和相关书籍。

在下一章，我们将把重点放在 Qt IDE 方面，其中有很多章节和内容都会与本章的基础 知识和技能相关，大家可以对照参考 学习，效果更佳。

第 4 章 Qt 4 集成开发环境

本章重点

- 了解常见的 Qt4 的 IDE
- 掌握 Qt Creator 的安装和配置
- 掌握 Eclipse 与 Qt4 结合使用的安装步骤和配置方法
- 掌握 Visual Studio 与 Qt4 结合使用的安装步骤和配置方法

4.1 常见的 Qt IDE

现在支持 Qt 的 IDE（集成开发环境，以下简称 IDE）有很多种，其中能够像 Qt 一样跨 平台使用的主要有 Qt Creator、QDevelop 、Eclipse 、MonkeyStudio、 Code::Blocks 等。在 X11 平台上还有老牌的 Kdevelop。在 Windows 平台上使用比较多的则主要是 Microsoft 的 Visual Studio 系列以及 Eclipse。在 Mac 上主要是使用 XCode。

在上述 IDE 中，Qt Creator 是最值得推荐的。因为它是 Nokia 官方推出的一款跨平台 开源 IDE，具有界面简洁、操作容易、与 Qt 结合完美等优点。Qt Creator 唯一可以被拿出 来指摘的就是它的资历尚浅，许多 功能尚不能做到稳定的实现。但是它的冲击力和前瞻性无 疑是更强的。

在 Qt 4.5 版以前，QDevelop 是很多开发者的首选 IDE，它很容易上手，是初学者的最 佳选择之一。但是自进入 2009 年以后，它的开发进度变得缓慢起来，甚至到笔者写作之 时，它还没有对 Qt 4.5 版提供正式的支持，这也导致了大量的使用者转而使用 Qt Creator。

Eclipse 是著名的开源 IDE，它的扩展性好，背后有业界巨头和数量众多的开源社区支 持，能够以插件形式支持 Qt 开发，虽然还不是很理想，但值得信赖 。

KDevelop 不能跨平台，一般只能在 X11 平台上使用，并且易用性不是很强。

Monkey Studio 是一款很有潜力的开源跨平台 IDE，已经被收录进众多 Linux 发行版软 件仓库中，但现在使用的人还比较少 。

Code::Blocks 也很优秀，并且其界面形式和操作方法与 MS 的 Visual Studio 很相似， 但它的安装配置很复杂，对中文的支持也不够理想，这导致 目前使用它的人数不是很多。

在 Windows 的平台上，使用 Microsoft 出品的 Visual Studio 系列与 Qt 集成开发程序 是非常好的组合，可以说是集合了两者的长处，比如 支持控件拖拉，编译调试方便，支持代 码的 IntelliSense 等等。

抛开与 Qt 集成使用的要求，目前业界同行里面使用 MS Visual Studio 6.0 版的还是 占多数，但是使用 2005 版和 2008 版的人数正在快速上升中，而使用 2002 和 2003 版的则 非常少（它们对标准 C++支持不理想，并且与 VS 6 系列不甚兼容）。考虑到一点，就是从 Qt4.5 开始，官方宣称不再支持 VS 6.0 版，所以我们在 Windows 平台上最好选择 2005 和 2008 版。

在 Mac OS X 上，主要使用 XCode、Qt Creator 和 Eclipse 作为 IDE。

在 S60 平台上，Qt + Carbide.c++ 2.x 是最为常见的组合。由于 Carbide.c++是 在 Eclipse 的基础上扩展的，所以掌握好 Eclipse 与 Qt 的组合使用是很有必要的。

基于上面的分析，在下面的章节中，我们将重点向大家介绍 Qt Creator、Eclipse 以 及 MS Visual Studio 系列与 Qt 4.5 的配置使用。

4.2 Qt Creator

如果说 Qt Creator 是当前最为耀眼的 Qt IDE，恐怕没有多少人会提出不同意见，目前 在全世界的 Qt 各大论坛里，关注和使用 Qt Creator 的朋友比比皆是，一时间好像其它的 IDE 都消失了一样，由此可见它的影响力。

本小节将简要的介绍 Qt Creator，在本书的第 12 章中将详细的讲解 Qt Creator 的使用。

4.2.1 简介

Qt Creator 是 Qt 被 Nokia 收购后推出的一款全新的跨平台开源 IDE，是 Qt SDK 的组 成部分之一，专为 Qt4 开发人员的需求量身定制。Qt Creator 的设计目标是使开发人员能 够利用 Qt 这个应用程序框架更加快速及简易的完成任务。由于捆绑了最新 Qt 库二进制软 件包和附加的开发工具，并作为 Qt SDK 的一部分，Qt Creator 在单独的安装程序内提供 了进行跨平台 Qt 开发所需的全部工具。

Qt Creator 是很年轻的，它于 2008 年 10 月的 Qt Developer Days 上被宣布(这项计划 代号为 Greenhouse)。该项目的技术预览版在 2008 年 10 月 30 日公布。最后在 2009 年 3月 3 日正式发布（连同 Qt 4.5），并提供 LGPL 许可的源代码。

4.2.2 主要特点

Qt Creator 有如下功能特色：

- 语法标识和代码完成功能的编辑器
- 项目生成精灵 ：允许用户生成控制台应用程序、GUI 应用程序、或 C++函式库的专案。
- 整合图形界面构建器 Qt Designer，能够用拖拉的方式将 Widget 排放在接口上，支持版面配置，支持信号与槽编辑。整合帮助文件浏览器 Qt Assistant
- 集成版本控制器，如 git、SVN
- 提供 GDB 和 CDB 侦错程式图形界面前端
- 默认使用 qmake 构建项目，也可支持 CMake 等其它构建工具
- 轻量级的开发环境
- 使用 g++ 作为编译器 4.2.3 安装与配置

Qt Creator 的安装时非常容易的，一般使用 SDK 或 Nokia 提供的独立安装程序来安 装。首先登录到 Qt Software 的官方网站，然后像图 4-1 所示那样选择适合你的平台的对 应安装程序下载并安装，安装过程一路选择缺省设置即可。

小贴士：最近经常有朋友在网上提问，说自己安装了 Qt Creator，为什么还不能开发 Qt 应 用程序，经常报错。这是由于还没有安装好 Qt 的缘故，没有安装好 Qt，当然就无法开发 Qt 应用程序了。所以，我们还是推荐大家尽量采用 SDK 的方式安装，这样就把你需要的组 件一次

安装齐了，省心又省力。

Qt Creator IDE

Qt® Creator 是全新的跨平台集成开发环境，与Qt 配合使用。

二进制软件包

下载用于 Windows 的 Qt Creator 1.2.1 二进制软件包 (26Mb)
下载用于 Mac 的 Qt Creator 1.2.1 二进制软件包 (52 Mb)
下载用于 Linux/X11 32位 的 Qt Creator 1.2.1 二进制软件包 (31 Mb)
下载用于 Linux/X11 64位 的 Qt Creator 1.2.1 二进制软件包 (37 Mb)
下载用于 Linux/X11 gcc3 的 Qt Creator 1.2.1 二进制软件包 (27 Mb)

源软件包

下载 Qt Creator 1.2.1 源软件包 (7 Mb)

图 4-1 Qt Creator 下载界面

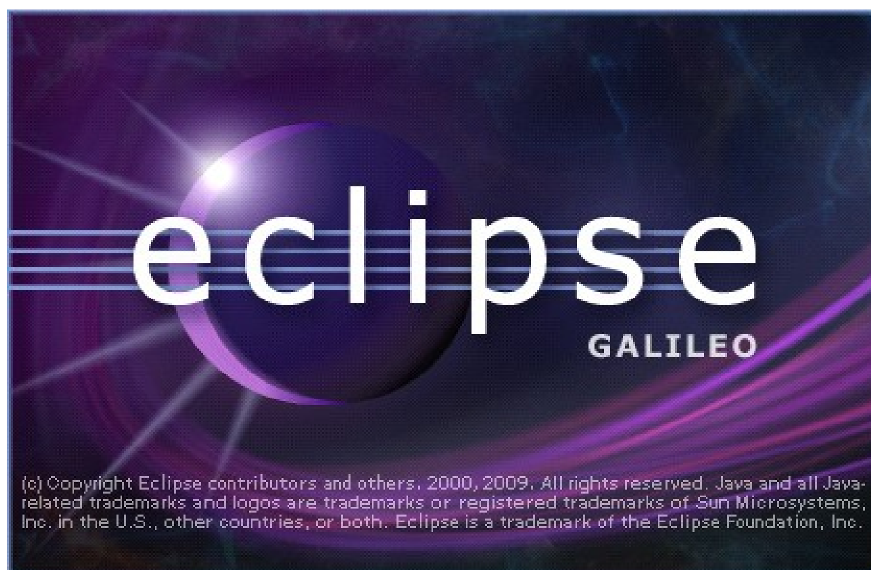
4.3 Eclipse

Eclipse 和 Qt 的结合起来使用，也是很常见的形式，Nokia 专门为此开发了插件。 Eclipse 的官方网站是：http://www.eclipse.org/，在上面可以很容易的下载到各种你需 要的版本，目前的最高版本为 3.5 Galileo。

4.3.1 简介

Eclipse 是著名的跨平台的开源的集成开发环境（ IDE）。Eclipse 的本身只是一个框架平台，最初主要用来 Java 语言开发，但是众多插件的支持使得 Eclipse 拥有其他功能相 对固定的 IDE 软件很难具有的灵活性。许多软件开发商以 Eclipse 为框架开发自己的 IDE，比如 Nokia 的 Carbide.c++（后面我们还会讲到它），现在它已经发展成为支持其他多种计 算机语言如 C++和 Python 的优秀 IDE。

Eclipse 启动后的画面如图 4-2 所示。



4.3.2 主要特点

Eclipse 的主要特点如下：

- 良好的扩展性

  图 4-2 eclipse 启动画面

  这要归功于 Eclipse 首创的插件机制，Eclipse 的设计思想是：一切皆插件。Eclipse核心很小，其它所有功能都以插件的形式附加于 Eclipse 核心之上。Eclipse 基本内核包 括：图形 API (SWT/Jface)， Java 开发环境插件（JDT），C/C++环境插件插件（CDT），开发框架环境（PDE）等。

- 强大的支持

  Eclipse 最初是由 IBM 公司开发的替代商业软件 Visual Age for Java 的下一代 IDE 开 发环境，2001 年 11 月贡献给开源社区，现在它由非营利软件供应商联盟 Eclipse 基金会（Eclipse Foundation）管理。众多的业界巨头对 Eclipse 非常看好，并且持续投入巨资以支持研发和管理。

  除了商业公司的青睐，Eclipse 还得到来自世界各地的爱好者组成的众多开源社区的 支持，几乎每天都有新的插件和功能产生，而 Eclipse 基金会的管理也非常有序，基于 Eclipse 的成功应用屡见不鲜，这些都使得 Eclipse 成为最受欢迎的优秀 IDE 之一。

- Qt 工程可以跨平台使用

  Eclipse 虽然不能解析 qmake 文件，但是使用它建立的同一个工程可以在 Windows、Linux 和 Mac OS X 使用。这就大大简化了移植应用程序的过程，与 Qt 的理念不谋而合。

- 代码提示功能突出

  与 Qdevelop 等其他 IDE 比较起来，Eclipse 没有使用 CTags 作为代码提示工具,因而在进行代码提示的时候速度更快,CPU 资源占用更少.并且 Eclipse 中的提示内容更加的准确和 完整。

Eclipse 也有一些不足之处，主要是两点。一是耗费系统资源较多，这与它的机制和 Java 的运行效率有关；二是对 Qt 的支持还不是很完善，这与它的设计理念有关，它 的初衷 只是提供一个可扩展的框架，许多的功能还是要其他厂商或开发者自己完善。

具体使用时的感觉就是用 Eclipse 开发程序，机器要有较好的配置，内存一定要多一 些，否则程序的运行速度将比较缓慢，开发效率不高。

总的来说，Eclipse 是一个很好的 Qt IDE，它也是 qtsoftware 官方网站上推荐的 IDE 之一。良好的扩展性以及众多业界巨头的和开源社区的支持，都使 Eclipse 值得信赖和推 荐。

4.3.3 安装与配置

Eclipse 本身的安装比较容易，但与 Qt 结合起来就需要一定的步骤了，尤其是在 Windows 平台上面。一般很少有人在 Linux 平台上使用 Eclipse 开发 Qt 程序，这主要是由 于如果采用编译安装的方法的话，有太多的依赖问题需要解决，而如果使用发行版自带的 Eclipse 的话，又存在与 Qt 版本的对应问题。在 Mac OS X 上同样存在类似的问题，开发者 一般使用 Xcode。所以使用 Eclipse 一般是在 Windows 这个单一平台上，并且是选择 Windows XP 以上的版本。我们就以 Windows XP SP2 中文版和最新的 Qt 4.5.2 为例，向大 家介绍如何使用 Eclipse 开发 Qt 应用程序。

首先看看需要准备那些软件包，表 4-1 描述了需要安装的软件包和下载地址。

表 4-1 需要的软件包

软件包	说明	下载地址
JRE	Java 运行环境	http://java.sun.com/javase/downloads/
MinGW	C++编译器和调试器	http://sourceforge.net/
Qt OpenSource	Qt 库	http://www.qtsoftware.com/downloads-cn
Eclipse	支持 C++开发的版本	http://www.eclipse.org/downloads
Qt Eclipse Integration for C++	Qt 与 Eclipse 的集成工具	http://www.qtsoftware.com/developer/eclipse-integration

安装过程如下：

第 1 步，确定安装顺序

由于 Eclipse 需要 JRE 才能运行，而 Qt Open Source 也依赖于 MinGW，所以我们按照下面的顺序安装软件包：JRE、MinGW、Qt Open Source 、Eclipse、Qt Eclipse Integration for C++。

第 2 步，安装 JRE

这里需要 JRE1.5 以上，我的选择是直接下载安装 JDK，其中包含了 JRE。安装方法可 以参考第 2 章，一路点击【Next】按钮，只是在设置安装路径时，不要选择带有空格和特 殊字符的就好。

第 3 步，安装 MinGW

MinGW 的安装也与第二章的方法相同，需要安装 DevCpp，或者选择在下面安装 Qt 时， 自动由 Qt 安装程序从网络上下载并安装，而不要自己下载安装最新的 5.1.4 版 MinGW。

第 4 步，安装 Qt 库

Qt Open Source 版可以选用 SDK，这样就不用先安装 MinGW 了。也可以选择框架，如 图 4-3 所示，在 qtsoftware 网站上，选择“下载用于 Windows 的 qt 库 4.5（60 Mb）”。 下载后的安装方法请看第 2 章。

Qt: 仅下载框架

如果不需要下载完整的 SDK，仅下载 Qt 框架的源软件包即可。你可以在安装过程中选择LGPL或GPL。

- 模块化 Qt 库
- 集成的 Qt 开发工具

下载用于 Windows 的 Qt 库 4.5 (60 Mb)

下载用于 Linux/X11 的 Qt 库 4.51(60 Mb)

下载用于 Mac 的 Qt 库 4.5 (60 Mb)

下载用于嵌入式 Linux 的 Qt 库 4.5 (50 Mb)

下载用于 Windows CE 的 Qt 库 4.5 (50 Mb)

图 4-3 下载 Qt 库

第 5 步，安装 Eclipse

去网站上下载 Eclipse，由于 Eclipse 项目众多，它的网站上下载列表项变得很长，让人困惑，如图 4-4 所示，我们要下载的包是那个"Ecilpse IDE for C/C++ Developers(79 Mb)"。你下载到的文件名字类似于 eclipse-cpp-galileo-win32.zip，是一个压缩包。解 开它后，只要你安装过了 JRE，就可以运行了。

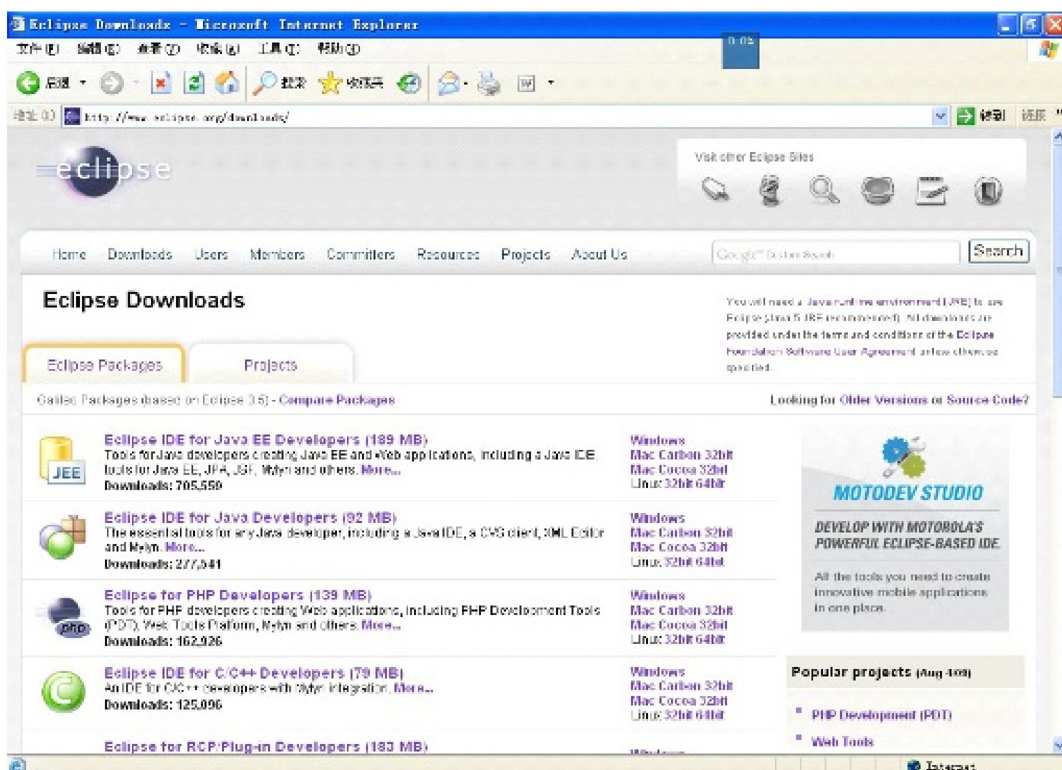


图 4-4 下载 ecilpse&CDT

接下来把它解压缩到你的目录中，为方便起见，可以把它解到某个盘符的根目录下，比如 C 盘，因为它会创建完整的路径，如图 4-5 所示。

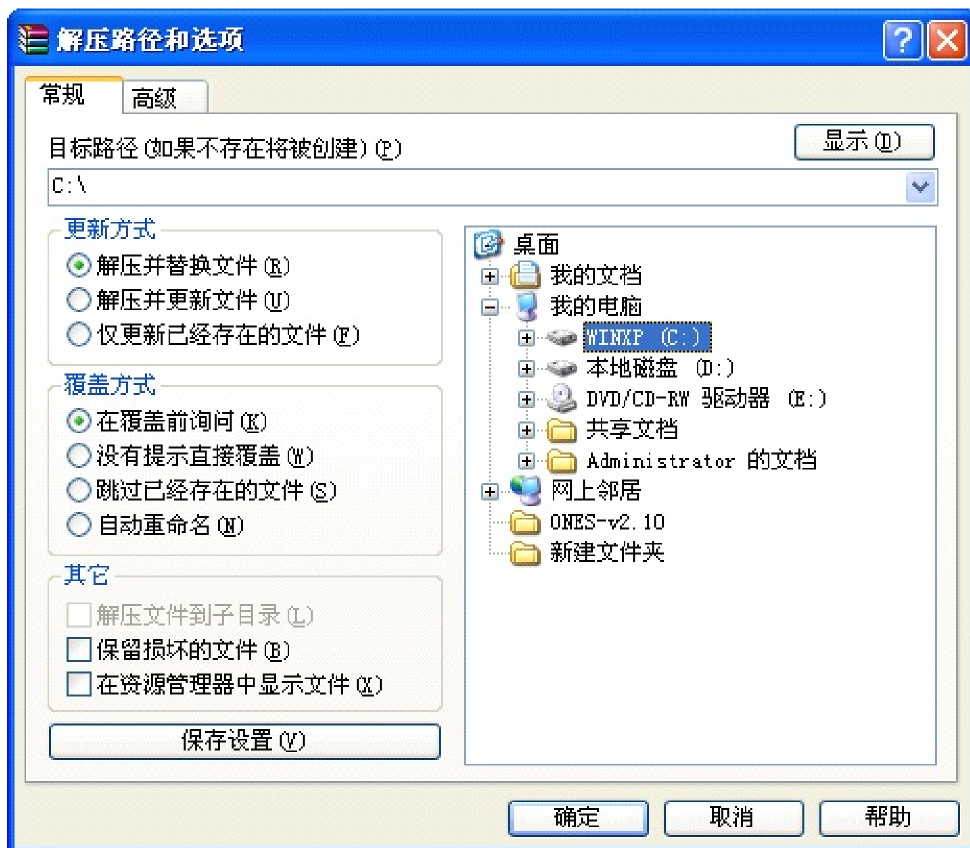


图 4-5 解压缩到根目录

这样解压缩后，我的情形是在 C 盘根目录下建立了一个 C:\eclipse 目录，如图 4-6 所 示，里面展开了所有内容，只要你前面安装了 JRE，这时候双击 eclipse.exe，就可以运行 了。

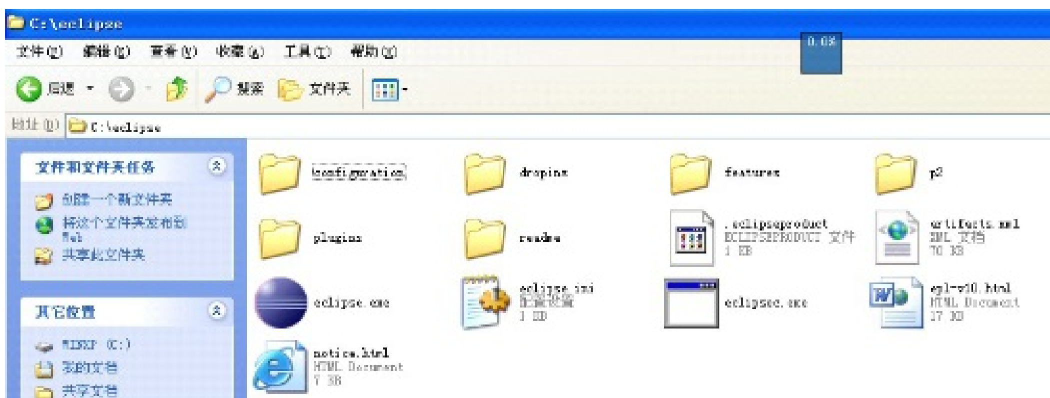


图 4-6 安装 eclipse 完毕

第 6 步，安装 Qt Eclipse Integration for C++

先到 Qt Software 网站上下载安装程序，情形如图 4-7 所示，我们需要下载 Windows 版的。



platform	specification	usage guidelines	built with version	download
Windows	Win32	All editions	Qt 4.5.2	Download
Linux (x86 32 bit)	gcc 4	All editions	Qt 4.5.2	Download
Linux (x86 64 bit)	gcc 4	All editions	Qt 4.5.2	Download
Linux (source 32 bit)	-	LGPL	-	Download
Linux (source 64 bit)	-	LGPL	-	Download

图 4-7 下载 Qt Eclipse Integration for C++

然后开始安装，中间过程基本都是一路选择默认。需要注意的是，选择 Eclipse 时需 要选择它的根路径，而 MinGW 则需要指定它的 bin 目录。如果你是采用 SDK 安装的 Qt 库， 那么你需要如图 4-8 所示进行设置。

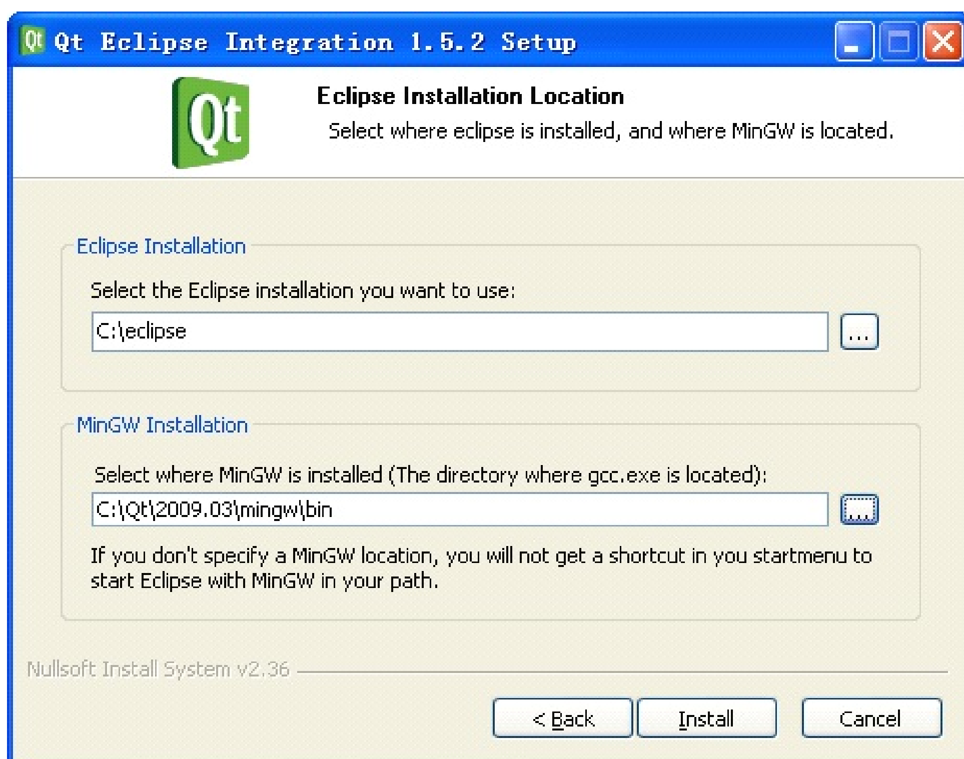


图 4-8 设置 eclipse 和 MinGW 的路径（SDK 方式安装 Qt 库）

如果采用框架方式安装，则你的设置应该如图 4-9 所示。

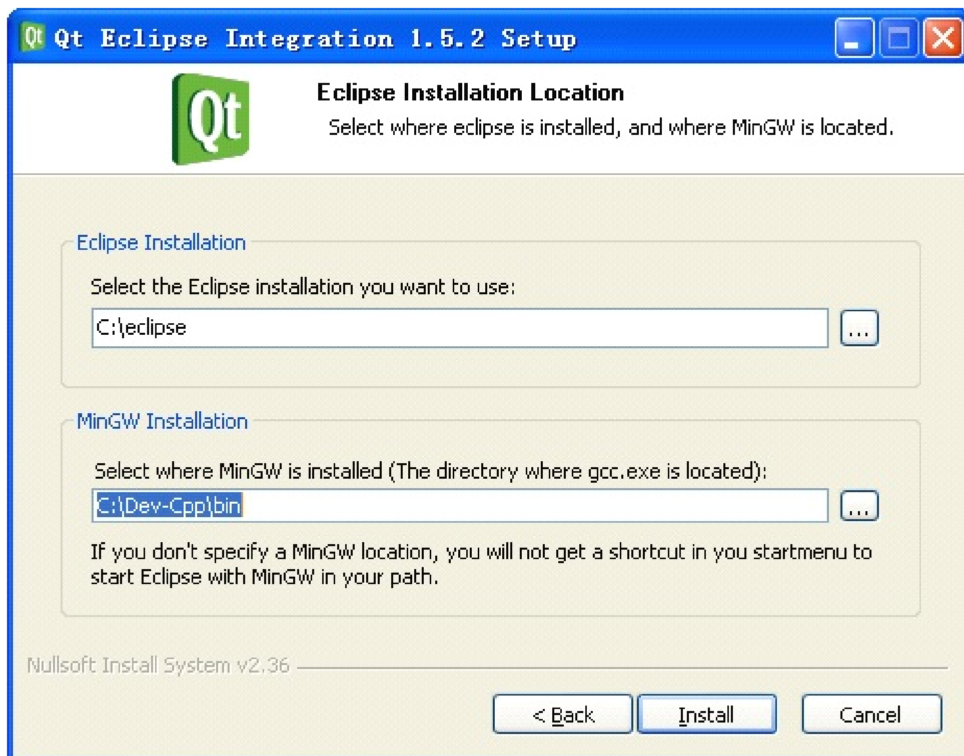


图 4-9 设置 eclipse 和 MinGW 的路径（框架方式安装 Qt 库）

接下来，就是一路按下【Next】按钮直至安装完成。 至此，Qt 和 Eclipse 的安装就完成了。

4.3.4 使用要领

1.软件的启动

你可以选择如下方式启动 Eclipse，依次点击菜单【开始】->【程序】->【qt eclipse integration】->【开始】，Eclipse 在弹出启动画面后，将进入如图 4-10 所示的主界面。

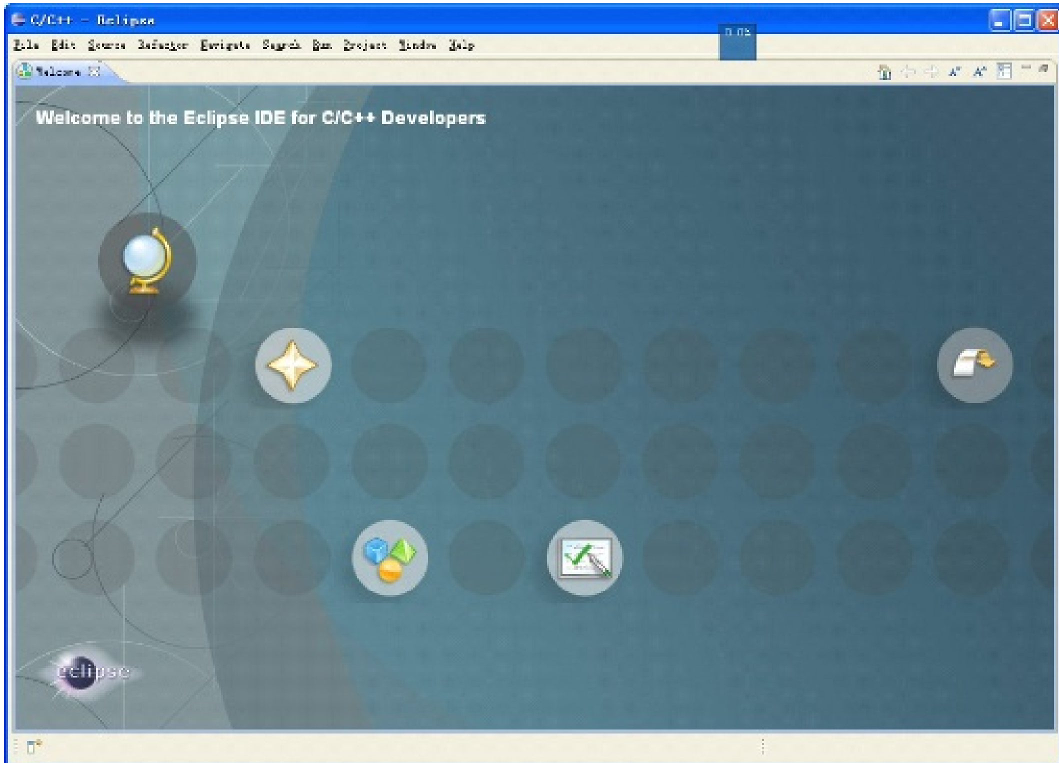


图 4-10 Ecilpse 成功启动

2.设置默认工作目录

在第一次启动 Eclipse 时，软件会要求设置默认工作路径，如图 4-11 所示在其中填入 你的路径即可。

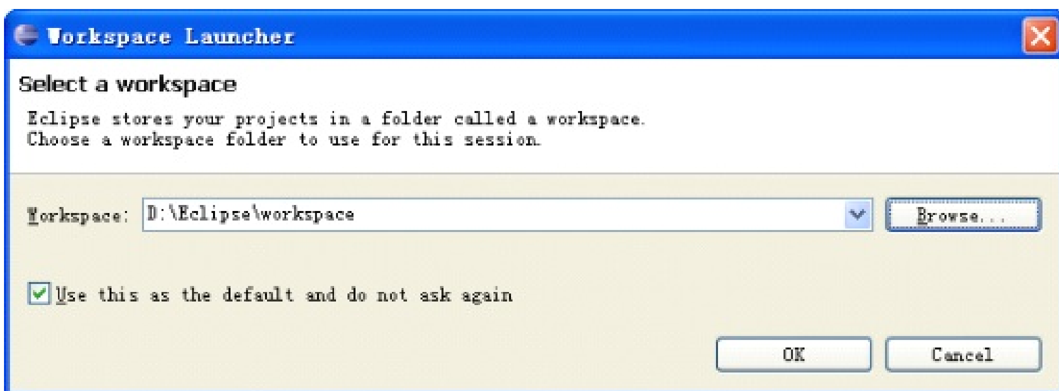


图 4-11 设置默认工作目录

小贴士：设置完默认工作目录之后，系统就不会再提出此要求了。但日后要想更改的话， 却找不到设置这一项的地方。因为 Eclipse 在第一次运行时才会去 plugins 目搜索插件， 之后就不再找了。而 Eclipse 不知道是出于什么原因，没有提供修改的方法。经过摸索， 解决方法有两种。

一种就是删除 Eclipse 的 configuration 目录，让 Eclipse 以为还是第一次启动，它 就会去搜索 plugins 找 Qt 插件了。

第二种就是加参数运行 Eclipse。在命令窗口下执行 eclipse.exe -clean 命令，Eclipse 就会清除配置文件然后重新搜索插件配置。

3.如何安装调试库

如果采用 SDK 安装 Qt 库，就已经安装了 debug 和 release 的动态库；如果采用框架方 式安装 Qt 库，默认是不安装调试库的，如需运行 Debug 模式，可以进入到 Qt 的 bin 目录 里面，输入如下命令即可编译调试库。

```
qtvars.bat compile_debug
```

或者依次点击【开始】->【程序】->【Qt by Nokia v4.5.2(OpenSource)】->【Qt4.5.2(Build Debug Libraries)】，这需要几个小时的编译时间和大约 10 个 G 的硬盘空 间，要有耐心，还要有足够的硬盘容量。

4.如何建立 Qt 工程

在主菜单上如图 4-12 所示依次点击【File】->【New】,就会出现多种可供选择的 Qt 工程类型，根据你的想法进行选择即可 。

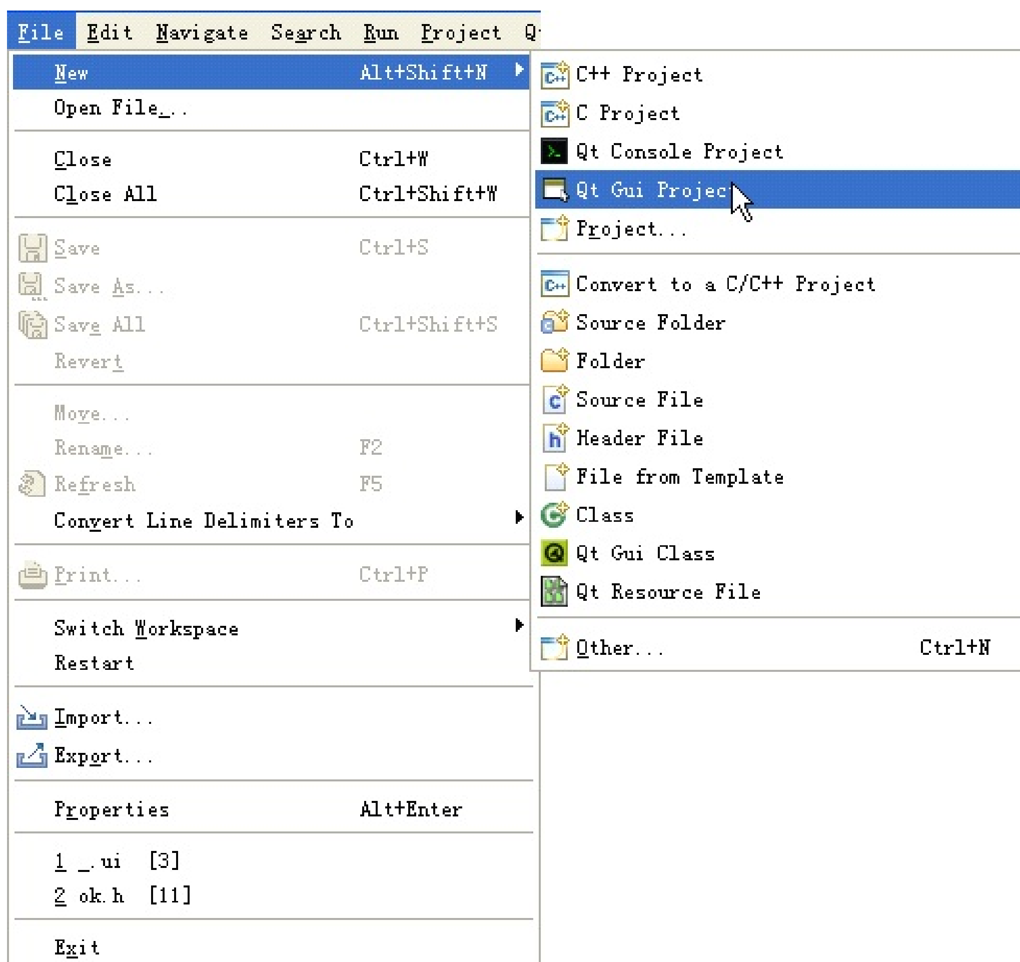


图 4-12 新建 Qt 工程

比如选择 Qt Gui Project，然后在出现的界面上放置一些常见控件，如图 4-13 所示， Qt Designer 已经无缝的集成到了 Eclipse 中。

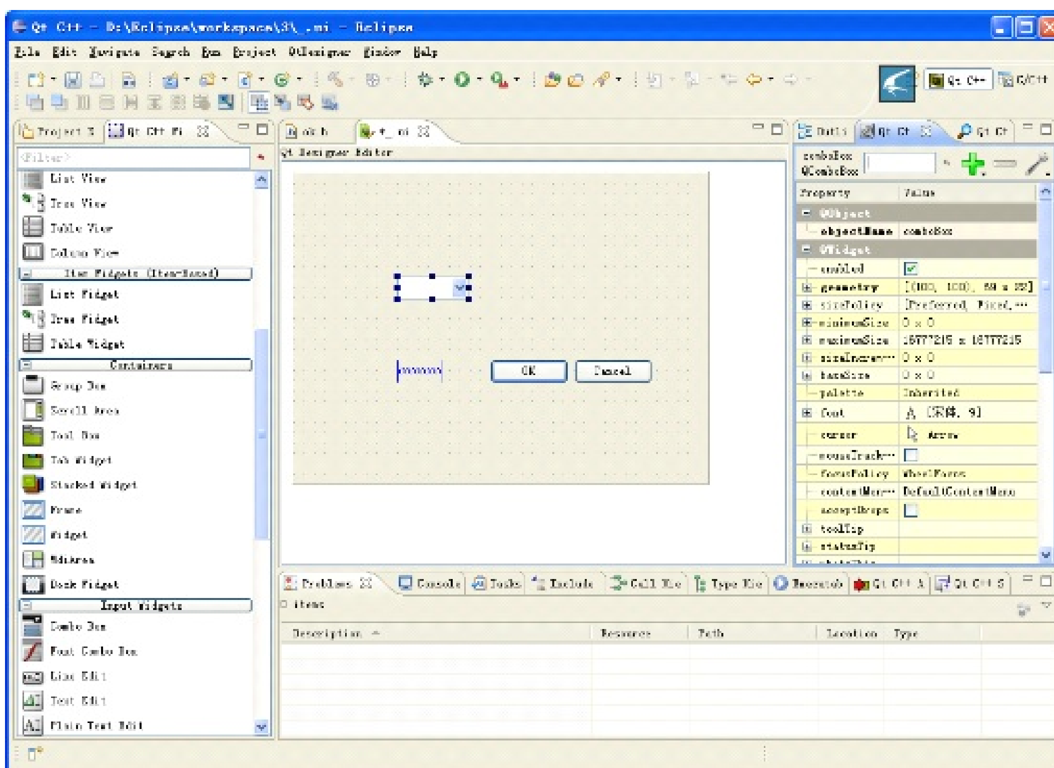


图 4-13 Eclipse 中集成 Qt Designer 4.4Visual Studio 2008（含 Express 版）与 Qt 4.5（含 4.5 的所有版本）

这里我们以 MS Visual Studio 2008 版为例，说明与 Qt 4.5 的集成使用方法，该方法 也适用于 Express 系列。VS 2005 与 Qt 4.5 的集成配置方法与此类似，只是在安装 VS 2005 时，需要打上 SP1 的补丁。我们更为推荐使用 VS 2008。据 Qt Software 透露的消息，自 Qt 4.6 开始，Qt 将提供针对 MS Visual Studio 系列 的直接安装版本，这一举措将大大简化安装配置的过程。但是，该版本内置的功能不一定能 够满足开发者的需求，所以本文介绍的编译安装方法是具有普遍意义的，希望大家能够用心 掌握。

下面以 Windows XP SP2 中文版为例，介绍如何把这两者集成使用。

第 1 步，安装 MS Visual Studio 2008

这一步里面，需要注意的有两点。一是可以选择英文版或者是中文版，在整个的安装 过程中并没有大的差异。使用中文版时，无论是 VS 2008 还是 2005 都需要安装 VS 的 SP1 包，否则会出现错误，导致安装失败。笔者个人喜欢使用英文版；二是安装时，最好选择完 全安装的方式，当然不会用到的语言和模块比如 VB 和 C#等就可以不选，VC 的所有组件要 全部安装上去。在网上好多朋友安装时出现了一些问题，很多都与 VS 安装时没有安装完全 有关。

第 2 步，安装 Qt 4.5 的源码

在官方网站的下载地址上下载 Qt 的 SDK，采用默认设置安装（不熟悉的朋友可以参阅第 2 章），注意版本是随时可能更新的，官网地址是：http://www.qtsoftware.com/downloads。我们这里选择 LGPL/Free Downloads，然后选择 qt-win-opensource-src-4.5.0.zip 解压，假定解压到 E:\qt-win-opensource-src-4.5.0，注意解压路径不要包含空格、中文名称以及其它特殊字符，解压后目录层次如图 4-14 所示。



图 4-14 解压后目录层次

第 3 步，配置环境变量 主要是设置两个环境变量，使用命令行方法如下：

```
set QTDIR=E:\qt-win-opensource-src-4.5.0
set path=%path%;%QTDIR%\bin
```

使用图形化设置的方法如下：

依次点击【我的电脑】->【属性】->【高级】->【环境变量】，弹出【环境变量】设 置对话框，如图 4-15 所示。

然后点击【新建】按钮，弹出【新建用户变量】对话框，在其中的输入框中填写："变量名" --- QTDIR "变量值" --- E:\qt-win-opensource-src-4.5.0



图 4-15 添加 Qt 的用户变量

与上面的步骤相似，修改 PATH 环境变量方法如下：

如图 4-16 所示，选择【PATH】变量，然后点击【编辑】按钮，在【变量值】的最后面 输入：

```
;%QTDIR%\bin
```

注意最前面有一个分号（不是中文的分号，切记）。

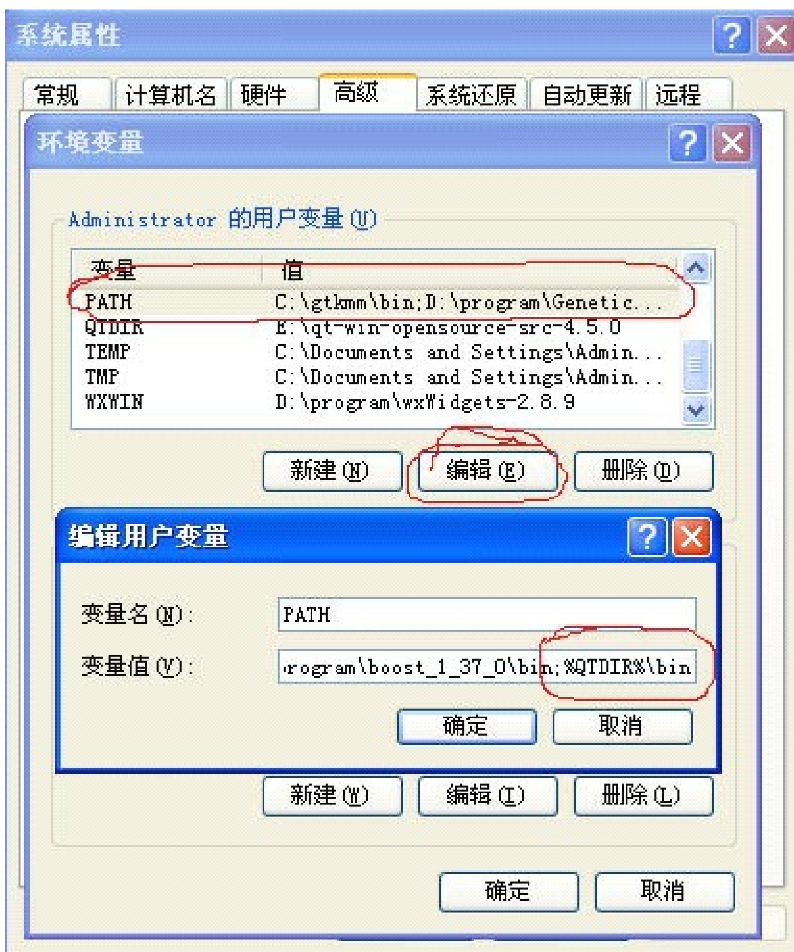


图 4-16 编辑 PATH 变量

第 4 步，使用 VS 编译 Qt 源码

最好先为 Qt 的 SDK 准备一个目录，如我的是:E:\Qt\4.5SDK。注意将磁盘格式设置为 FAT 32 格式，NTFS 格式可能会对安装过程有影响。

然后就开始编译 Qt 4.5 的源码。

通过【开始】菜单 ->【Microsoft Visual Studio 2008】->【Visual Studio Tools】, 运行 VS 2008 命令行提示，中间过程如下：

```
C:\Program Files\Microsoft Visual Studio 9.0\VC&gt; cd c:\Program Files\Microsoft Visual
C:\Program Files\Microsoft Visual Studio 8\Common7\Tools&gt; vsvars32.bat
C:\Program Files\Microsoft Visual Studio 8\Common7\Tools&gt; e:
E:\&gt; cd e:\qt\4.5sdk\qt
E:\Qt\4.5SDK\qt&gt; configure -platform win32-msvc2008 -debug-and-release
E:\Qt\4.5SDK\qt&gt; nmake
```

使用目前主流机器配置，configure 这一步大概要用一小时。namke 大概要三个多小时。中间可能会有一些警告，可以不理会。

第 1 行和第 2 行，是配置 VC 的环境变量，如何知道环境变量已经配置好了呢？方法是在命令行下运行：

```
cl.exe /?
```

如果输出了 cl.exe 的帮助说明信息，则表示 vc 编译器环境变量设置成功。

小贴士：在命令行下，可以通过输入 set path 、 set include 和 set lib 等命令查 看 path、include、lib 环境变量的设置情况。

configure 命令主要是做两件事。

一是编译 qmake，并将编译好的 qmake.exe 拷贝到 bin 目录下（这就是要添加"%QTDIR%\bin "到 PATH 环境变量中的原因）。

二是生成 makefile 文件（包括编译 qt 的 makefile 文件以及 examples、demos、tools 等的 makefiles 文件）。

configure 的使用是需要仔细琢磨的，可以通过加 -h 参数来看具体有哪些选项，有些 选项前面加*表示默认是选中的，+号表示由系统来判断是否支持该选项。表 4-2 列出了常 用的 configure 命令的常用参数。

表 4-2 configure 命令的常用参数

编译 **shared** 版（默认编译为 **shared** 版，不需要设置），并且 编译 **debug** 和 **release** 两个版本	**-debug-and-release**
使用 vc2008（包括 express 版）	-platform win32-msvc2008
使用 qt 自带的 zlib、gif、libpng、libmng、libtiff、libjpeg	-qt-zlib -qt-gif -qt-libpng -qt-libmng -qt-libtiff - qt-libjpeg
编译数据库插件支持 sqlite、odbc（需要相应的 C/C++头文件 和库的支持）	-plugin-sql-sqlite -plugin-sql-odbc
不支持 qt3	-no-qt3support
无 mmx 指令集支持	-no-mmx
无 3dnow 指令集支持	-no-3dnow
无 sse 和 sse2 指令集支持	-no-sse -no-sse2
无 direct3d 支持（默认不编译 direct3d）	-no-direct3d
无 openssl 支持	-no-openssl
无 dbus 支持	-no-dbus
无 phonon 支持以及 phonon 向后兼容性支持	-no-phonon -no-phonon-backend
不编译 webkit 模块	-no-webkit
不支持脚本工具 scripttools	-no-scripttools
不生成 sln 以及 vcproj 文件，只生成 makefile	-no-dsp -no-vcproj

使用表中列出的参数，其对应的 configure 命令如下：

```
configure -debug-and-release -platform win32-msvc2008 -qt-zlib -qt-gif -qt-libpng -qt-lib
-qt-libtiff -qt-libjpeg -plugin-sql-sqlite -plugin-sql-odbc -no-qt3support -no-mmx -no-3d
-no-sse -no-sse2 -no-openssl -no-dbus -no-phonon -no-phonon-backend -no-webkit -no-
scripttools -no-dsp -no-vcproj
```

编译完成之后，最好进行清理以节省硬盘空间，在命令行下输入：

```
nmake confclean
```

这样清理完成之后，整个 qt 解压目录大小约为 800M 左右。

第 5 步，最后，将 Qt 路径添加到 VC 编译环境中。

依次打开【工具】->【选项】->【项目和解决方案】->【VC++目录】。 在包含文件一栏添加：

```
E:\qt-win-opensource-src-4.5.0\include\QtGui； E:\qt-win-opensource-src-4.5.0\include\QtC
E:\qt-win-opensource-src-4.5.0\include
```

在库文件一栏添加：

```
E:\qt-win-opensource-src-4.5.0\lib
```

将 Qt 安装路径添加到 PATH 系统环境变量中，例如：E:\qt-win-opensource-src-4.5.0\bin（这一步可以省略，在本文的前面已经设置好了）。

第 6 步，安装 Qt for VS 的插件

要使 VS 中能新建 Qt 的项目及增加相关菜单和支持拖拉控件，还要再安装一个 Qt for VS 的插件。

在 Qt 的官网上下载该插件，可能的名字为 qt-vs-addin-1.0.2.exe（以你下载到的版 本为准）。在安装插件的时候可能会报找不到 Qt 目录，不要管它，只要找准了你刚才编译 的 Qt 目录就可以了。

安装完之后，再次打开 VS2008，就能发现已经支持 Qt 了。 第 7 步，配置和使用 VS

最后需要做的一步是在 VS 中指定使用的 Qt 目录。如果你在 Windows 上安装了多个版 本的 Qt Source，在 VS 中你还可以选择使用哪个版本的 Qt 来编译你的应用程序。

打开【工具】->【选项】->【Qt】->【Builds】，新建或选择一个 Qt 版本。

另外，通过菜单【Qt】->【Open Solution from .pro File】，可以把一个原本 Qt 的非 VS 的项目转成 VS 的项目。

至此，Qt4.5 和 MS Visual Studio 的编译环境基本上配置完成了，可以进行开发了。

4.5 问题与解答

问：我使用 Qt 4.5 和 Visual Studio 2008。我在编译 Qt 4.5 时总有提示：没有包含:"windows.h"，然后就无法进行 configure，请指教是什么问题。

答：这通常是 Visual Studio 没有安装完全的缘故。你需要完整的卸载 Visual Studio 2008，再重新安装完全版本，然后再编译 Qt 就应该没有错误了。如果习惯的话， 使用英文版 Visual Studio 更好。

问：Qt4.5 + visual studio 2008 中文版的问题。我安装好后，打开 Qt 的 Example, 编译的时候报错误，错误提示如下：

```
&gt;------ Build started: Project: calculatorform, Configuration: Debug Win32 ------
&gt;UIC calculatorform.ui
&gt;Moc'ing calculatorform.h...
&gt;Compiling...
&gt;main.cpp
&gt;calculatorform.cpp
&gt;Generating Code...
&gt;Linking...
&gt;LINK : fatal error LNK1181: cannot open input file 'QtCored4.lib'
```

请帮忙解决。

答：从错误信息上推断，应该是安装过程出了问题。有几个关键地方需要注意，一是 安装 Visual Studio 时如果是中文版，要打上补丁 SP1；二是安装 Visual Studio 要完 整；三是不要安装 mingw，要使用 Visual Studio 编译 Qt；四是注意配置环境变量。五是 注意安装好集成插件。

问：可否实现在 Linux 下编译 Windows 环境下用 Qt + Visual Studio 2008 创建的工程？

答：这是可以实现的。步骤如下：

第 1 步，生成 .pro 文件。

第 2 步，在 pro 文件里面加宏 DEFINES += Linux 。

第 3 步，把整个工程拷贝到 Linux 机器上。

第 4 步，在命令行下面执行：

```
qmake -makefile xxx.pro
```

生成 Makefile 文件。

第 5 步，在命令行下面运行 make 命令，生成可执行文件。 问：Visual Studio 2005 每次调试都重新执行 moc

我的 Visual Studio 2005 没有修改任何文件, 但每次调试都会重新 moc, 非常麻烦。 不知是什么原因,请指教。

答：这个与 Qt 关系不大，是 Visual Studio 的缘故。通常是由于在你的工程中，有的 文件时间设置系统时间要靠后，就是晚了。重新调整一下时间即可。

问：Qt 4.5 与 Visual Studio 集成时候出错

我将 Qt 4.5 与 Visual Studio 2005 结合使用，在执行 nmake 后，安装了 qt-vs- addin-1.0.0.exe 这个软件。但是随后打开 VS 2005 时,出现如图 4-17 所示的提示信息。



图 4-17 提示信息

然后在 Visual Studio 2005 的界面上 Qt 菜单下面没有任何的下拉菜单内容，如图 4- 18 所示。不知道是什么原因，请帮助解决一下。



图 4-18 Qt 菜单式空白的

答：你使用的是中文版的 Visual Studio，所以你需要在安装 Visual Studio 时，同时安装上它的补丁 SP1，这样就不会出现图 4-17 所示的那个问题了，或者干脆使用英文版的Visual Studio。然后使用本章中的步骤把 Qt 与 Visual Studio 结合起来使用。有条件的 话，也可以使用 qt-vs-integration 这个系列的集成插件，它更为稳定一些。

问：我想在 Visual Studio 2008 中直接打开.ui 文件，而不是使用 Qt Designer，如 何办到？

答：这与你使用的集成插件版本有关。

到目前发布的 Qt 4.5.2 为止，如果在安装时使用的是 qt-vs-integration 就可以在 Visual Studio 2008 或者是 2005 中打开.ui 文件，但是 qt-vs-integration 这个软件不是 Open Source 的，需要有商业 license 才可以；如果使用的集成插件是 qt-vs-addin，这个 是 Open Source 的，但是在 Visual Studio 中就只能使用 Qt Designer 打开.ui 文件。

也就是说，在商业版本的 Qt 中，与 Visual Studio 结合起来使用，是可以无缝集成 的。而在 Open Source 版本的 Qt 中，与 Visual Studio 的集成使用是有一定限制的。

从 Qt Software 发布的 Qt Road Map 中可以推测，在 Qt 4.6 中这一情况有望得到改 善，即商业版和开源版 Qt 都可以与 Visual Studio 系列无缝集成。

4.6 总结与提高

本章主要介绍了目前 Qt4 的主流 IDE，其中最为常用的是 Qt Creator、Eclipse 和 Visual Studio 系列。如果需要在不同的平台间穿插使用，笔者建议考虑 Qt Creator 和 Eclipse，以 Qt Creator 作为首选。在单一平台 Windows 上，可以考虑 Visual Studio 系 列，这其中又以英文版的 Visual Studio 2008 最为推荐。这里再提一下 Qt Creator，越来 越多的朋友选择它是有道理的。Qt Software 官方出品、雄厚的资金和研发背景、 业界巨头 的力推、强有力的社区支持、跨平台并且是开源的，清爽的界面和便捷的操作方式、吸引人 的 GPL 和 LGPL 授权方式等等，Qt Creator 的强点实在是太多，简直是集万千宠爱于一身， 无论是初学者还是资深的专家，都很难找出拒绝它的理由。 在后面的章节中，还将对 Qt Creator 的使用做更为详尽的讲解，并且程序实例都是使用 Qt Creator 完成的，希望大家 能够重点掌握。

第 5 章 使用 Qt 基本 GUI 工具

本章重点

- 使用 Qt Designer（设计师）进行 GUI 设计
- 使用 Qt Demo 浏览 Qt 应用开发
- 使用 Qt Assistant（助手）获取在线文档与帮助

5.1 使用 Qt Designer 进行 GUI 设计

5.1.1 简介

Qt Designer，又被称作是 Qt 设计师，是一个所见即所得的全方位 GUI 构造器，它所 设计出来的用户界面能够在多种平台上使用。它是 Qt SDK 的一部分，也是最为重要的开发 工具之一。利用 Qt Designer，我们可以拖放各种 Qt 控件构造图形用户界面并可预览效 果。

通常一个 Qt Designer 的样子如图 5-1 所示。

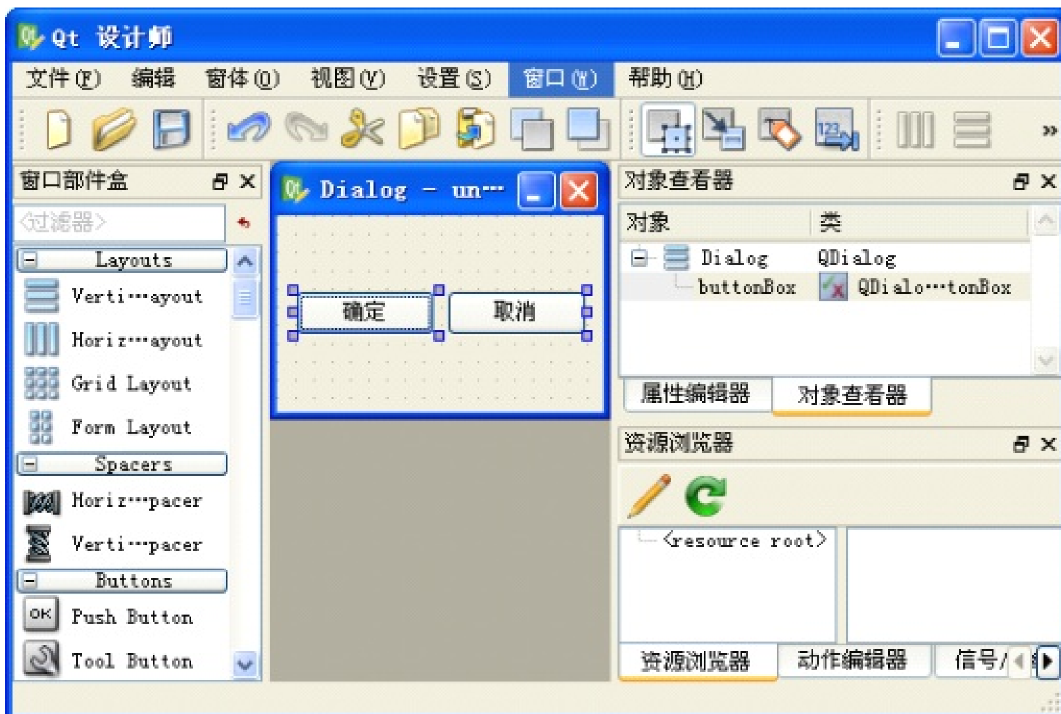


图 5-1 Qt Designer 的观感

使用 Qt Designer，开发人员既可以创建“对话框”样式的应用程序、又可以创建带有菜单、工具栏、气球帮助以及其他标准功能的 “主窗口”样式的应用程序。Qt Designer 提供了多种窗体模板，开发人员可以创建自己的模板，确保某一应用程序或某一系列应用程序界面的一致性。编程人员可以创建自己的自定义窗体，这些窗体可以轻松与 Qt Designer集成。

Qt Designer 支持采用基于窗体的方式来开发应用程序。窗体是由用户界面 (.ui) 文 件来表示的，这种文件既可以转换成 C++ 并编译成一个应用程序，也可以在运行时加以处 理，从而生成动态用户界面。Qt 的构建系统能将用户界面的编译构建过程自动化，使设计 过程更轻松。

Qt Designer 可以轻松的与许多常见的 IDE 集成 Windows 平台的商业许可证持有人可 以在 Microsoft Visual Studio® 内充分享受到 Qt Designer 用户界面设计所带来的便 利。在 Mac OS X 中，开发人员则可在 Apple's Xcode® 环境内使用 Qt Designer。另外， Nokia 还为跨平台的 Eclipse 集成环境开发了 Qt 的集成插件，以将 Qt Designer 及其他 Qt 技术嵌入到集成环境框架中。

5.1.2 启动并设置 Qt Designer

要运行 Qt Designer，在 Windows 下，可单击“开始”菜单中的“Qt SDK by Nokia v2009.3(Open Source) | designer”；在 UNIX 下，可在命令行终端中输入 designer 命 令；在 Mac OS X Finder 中，只需双击 designer 即可。以 Linux 系统为例，启动 Qt Designer 通常有 2 种方法：

1.程序组启动

一般的，如果你的 Qt 是采用发行版已经编译好的 Qt 包的话，安装完毕后就已经在程 序组中设置了链接项，一般是在【开发】或是在【编程】组中。以笔者的 Red Flag 为例， 依次点击【应用程序】→【开发】→【Qt】→【Qt4 Designer–Interface Designer】 后，Qt Designer 就会启动了。

2.命令行启动

如果你是自己采用编译源代码的方式安装 Qt，并且没有设置快捷方式的话，你可以从 命令行启动 Qt Designer。方法是进入到命令行方式，或者打开一个终端，进入你 Qt 安装 的目录，以笔者的 Red Flag 为例，进入 /usr/bin/目录，输入 ./designer-qt4，即可启 动 Qt Designer。

注意，不同的 Linux 发行版的菜单设置以及可执行文件命名可能会有所不同。 3.设置 Qt Designer启动 Qt Designer 后，首先需要需要对它进行设置。

Qt Designer 支持两种界面排列形式：一种是多个顶极窗口并列分布形式；一种是与Windows 上常见的 IDE 类似的单窗口（即多停靠窗锚接）形式。

多个顶级窗口的样子如图 5-2 所示，Qt Designer 的各个子窗口都作为独立的顶级窗口 排列在屏幕中，用户可以自由改变其位置和尺寸大小。

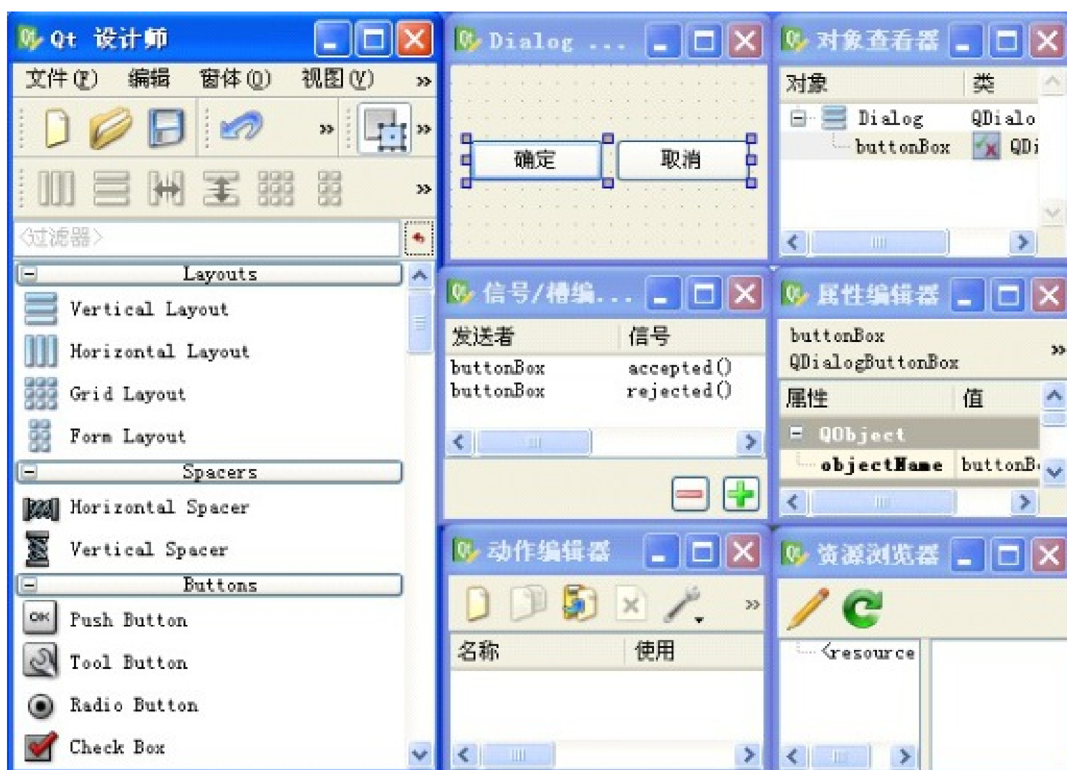


图 5-2 处于“多个顶级窗口”模式下的 Qt Designer

锚接窗口模式下的 Qt Designer 的样子如图 5-3 所示。

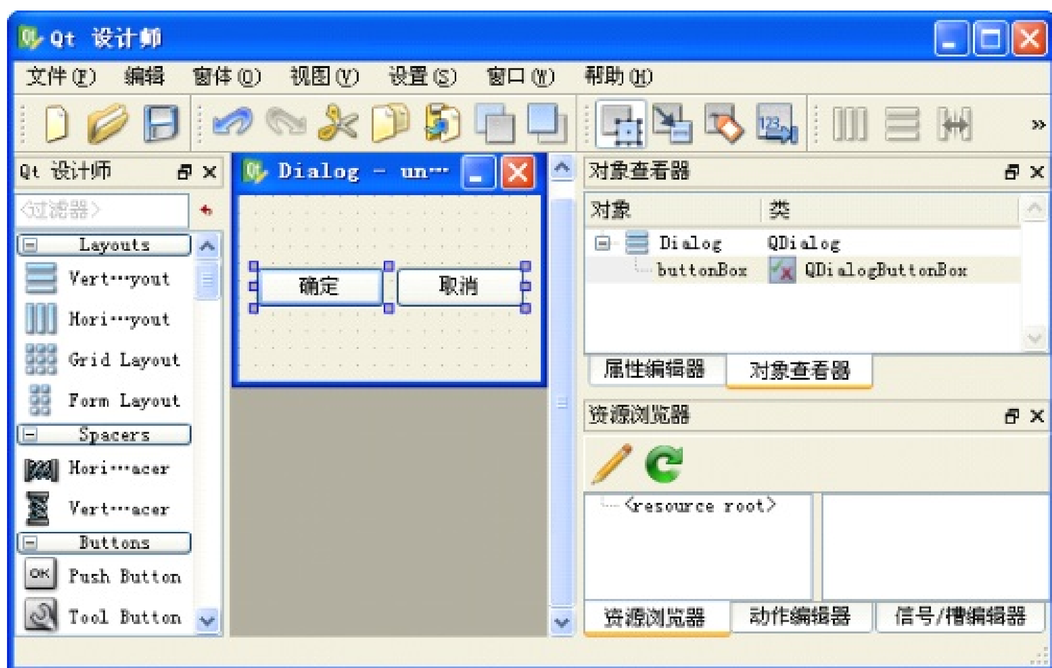


图 5-3 处于“锚接的窗口” 模式下的 Qt Designer

在 Qt4.5 版以前，Qt Designer 默认情况下以多个顶极窗口并列的形式显示，在 Qt4.5 之后，默认是锚接窗口的形式。这一改动受到了大多数开发者的欢迎，因为大家还是习惯使 用锚接窗口的界面布局形式。

如果你想变更 Qt Designer 的界面布局形式，在主菜单上依次选择【设置】→【属性】→【外观】→【锚接的窗口/多个顶极窗口】即可。

5.1.3 功能说明

1.界面布局

Qt Designer 主要由窗口部件盒、对象查看器、属性编辑器、资源浏览器、动作编辑器 和信号/槽编辑器组成，它们都是锚接窗口，通常排列在窗体的两侧，也可以定制它们的位 置。如图所示，界面中间是新建立的窗体。

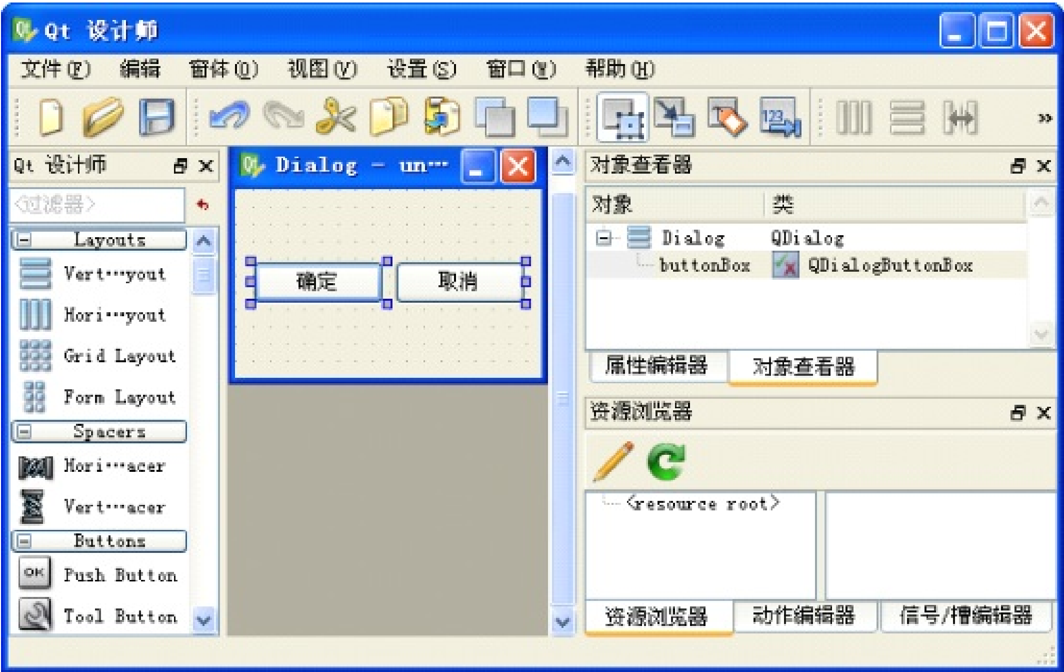


图 5-4 Qt Designer 的布局

2.功能项说明

表 5-1 示出了 Qt Designer 主菜单项的功能说明。

表 5-1 菜单项功能说明

菜单项	说明
文件	提供文件的新增、读取、存储、打印以及退出 Qt Designer 等命令
编辑	提供数据的剪贴、复制、查找，光标的跳转、代码补全；编辑窗口部件、编辑信号/槽、伙伴、Tab 顺序等命令
窗体	提供窗体属性设定、布局设置、调整 大小、预览等命令
视图	提供各种配置窗口、工具栏等的显示和隐藏设置等命令
设置	提供窗体属性设定，附加字体设定等命令
窗口	提供窗口的切换以及最小化等命令
帮助	提供调用 Qt Asssistant 等帮助命令

表 5-2 示出了文件工具栏功能说明。

表 5-2 文件工具栏功能说明

名称与图标	功能
	新建窗体
	打开窗体
	保存窗体

表 5-3 示出了编辑工具栏功能说明。

表 5-3 编辑工具栏功能说明

名称与图标	功能
	撤销上一次操作
	恢复操作
	剪切
	复制
	粘贴
	放到后面
	放到前面
	编辑窗口部件
	编辑信号/槽
	编辑伙伴
	编辑标签顺序

表 5-4 示出了窗体布局工具栏功能说明。

表 5-4 窗体布局工具栏功能说明

名称和图标	功能
	设置为水平布局
	设置为垂直布局
	设置为分裂器水平布局
	设置为分裂器垂直布局
	设置为栅格布局
	设置为表单布局
	打破布局
	调整大小;

3.主要部件的使用

窗口部件盒（Widget Box）

窗口部件盒是 Qt Designer 为使用者提供的窗口部件集合，根据你安装的 Qt 包的版本 和种类的不同，窗口部件盒中部件的种类也不尽相同，通常有 Layouts、Spacers、 Buttons、Items Widgets(Model-Based)、Item Widgets(Item-Based)、Containers、Input Widgets、Display Widgets 等大类。

以笔者使用的 Qt4.5.2 开源版为例，通常一个窗口部件盒的样子如图 5-5 所示

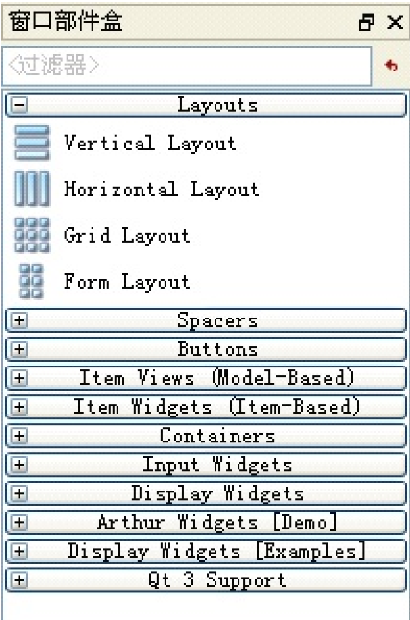


图 5-5 窗口部件盒

使用窗口部件盒的方法是用左键选中某一个部件，将它拖动到你的窗体屏幕上去，并 释放鼠标左键，这样就在界面上添加了这个部件。

属性编辑器（Property Editor） 属性编辑器是另一个非常重要的组件，当我们将界面上的窗体放置好后，就需要对各个组件的属性进行设置，通常一个属性编辑器的样子如图 5-6 所示。



图 5-6 属性编辑器

属性编辑器的使用也是非常简单的，使用者点击界面上的某一个部件，然后在属性编 辑器中设置属性即可，属性编辑器中属性栏的项目都是以英文来显示的，只要你对常用的计 算机编程中的英文词汇有所了解，理解它们的意思并正确的设置属性并不难。我们会在后面 结合具体实例为大家讲解。

对象查看器（Object Inspector）

对象查看器用来查看和设置窗体中的各个对象及其属性。

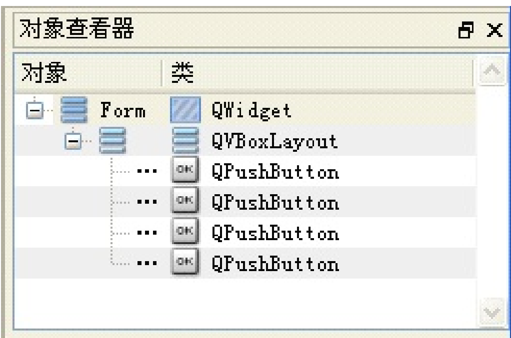


图 5-7 对象查看器

对象查看器通常与属性编辑器配合使用，一般是在设置好界面元素后，在对象查看器 中查阅各个元素的分布以及总体的布局情况，然后选中其中某个窗口部件，切换到属性编辑 器中编辑它的属性。

Qt Designer 还有几个重要的部件如资源浏览器、动作编辑器、信号 /槽编辑器等，我 们将在下面各节以及后面的各章中结合实例为大家介绍。

5.1.4 Qt GUI 设计基本流程

使用 Qt Designer 设计窗体十分简单,一般遵循如下的步骤。

第 1 步，启动 Qt Designer，选择要创建的应用类型。

启动 Qt Designer 后，将出现如图 5-8 所示的新建窗体对话框。



图 5-8 新建窗体对话框

Qt 4.5 默认为使用者提供了 3 大类模板供选择，即常见的窗口模板（templates\forms）、窗口部件盒自定义窗口部件。我们最为常用的是第 1 大类，其中包 括了对话框、主窗口和普通窗口部件等几种应用类型模板。这里我们以最后一项 Widget 为 例，选中它，再点击【创建】按钮，一个空白的 Widget 就创建好了,如图 5-9 所示。

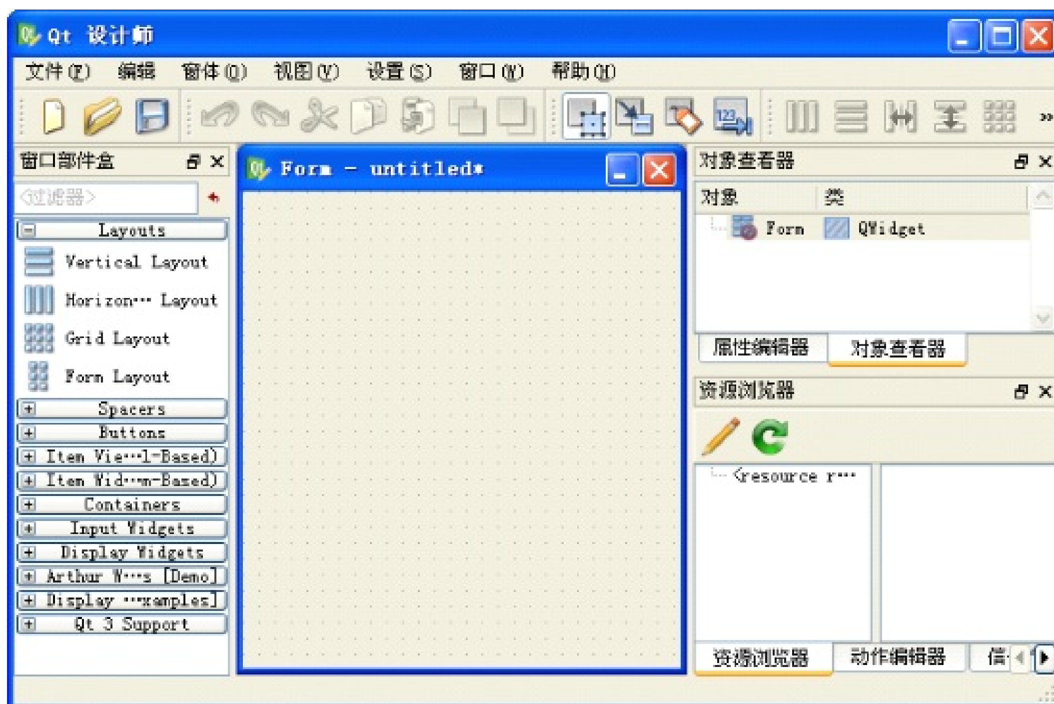


图 5-9 创建 Widget 类型界面

第 2 步，将控件从窗口部件盒拖到窗体上，然后使用标准编辑工具来选择、剪切、粘贴窗体并重新调整大小。以图 5-10 为例说明，我们从窗口部件盒里面，拖动出 2 个 Label、2 个 LineEdit、1 个 PushButton 和 1 个 Horizontal Spacer，大致排列一下放在窗体上。

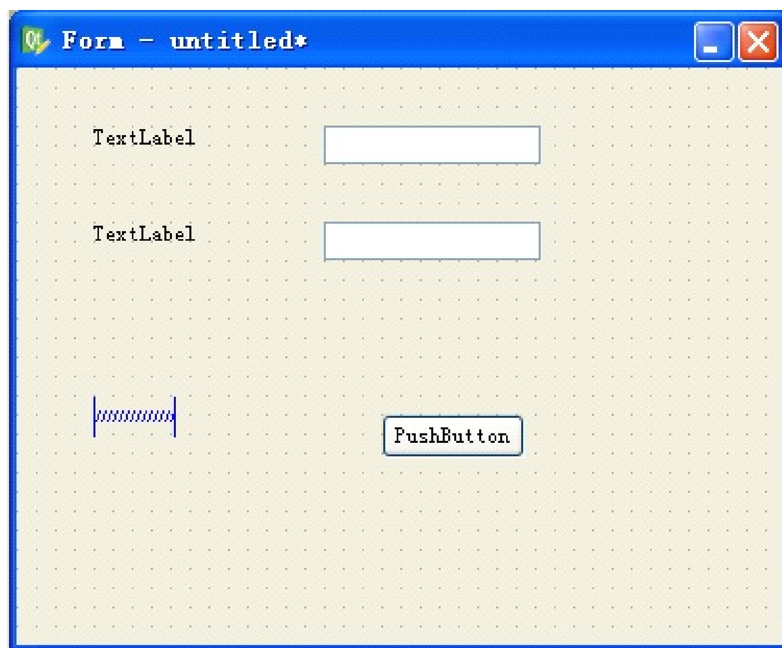


图 5-10 添加窗口部件

第 3 步，使用属性编辑器来更改窗体和每个控件的属性。

基本的属性包括窗口部件的 objectName、text 以及大小位置等。在表 5-5 中列出 了这几个窗口部件的常见属性设置，这里我仅仅举出 objectName 和 text 这两个最为 常用的属性，其它的属性设置与此类似。这样设置完成后的界面大致如图 5-11 所示的 样子。

表 5-5 窗口部件的属性设置

部件类别	**objectName**	**text(WindowTitle)**
Widget	myForm	布局示例
Label	label_name	姓名
Label	label_phone	电话
LineEdit	lineEdit_name	无
LineEdit	lineEdit_phone	无
Horizontal Spacer	horizontalSpacer	无
PushButton	pushButton_ok	确定



图 5-11 设置好部件属性的窗体

第 4 步，保存窗口设置，按下 Ctrl+Shift+S 组合键，在弹出的如图 5-12 所示的【窗 体另存为】对话框中输入文件名，然后点击保存按钮。

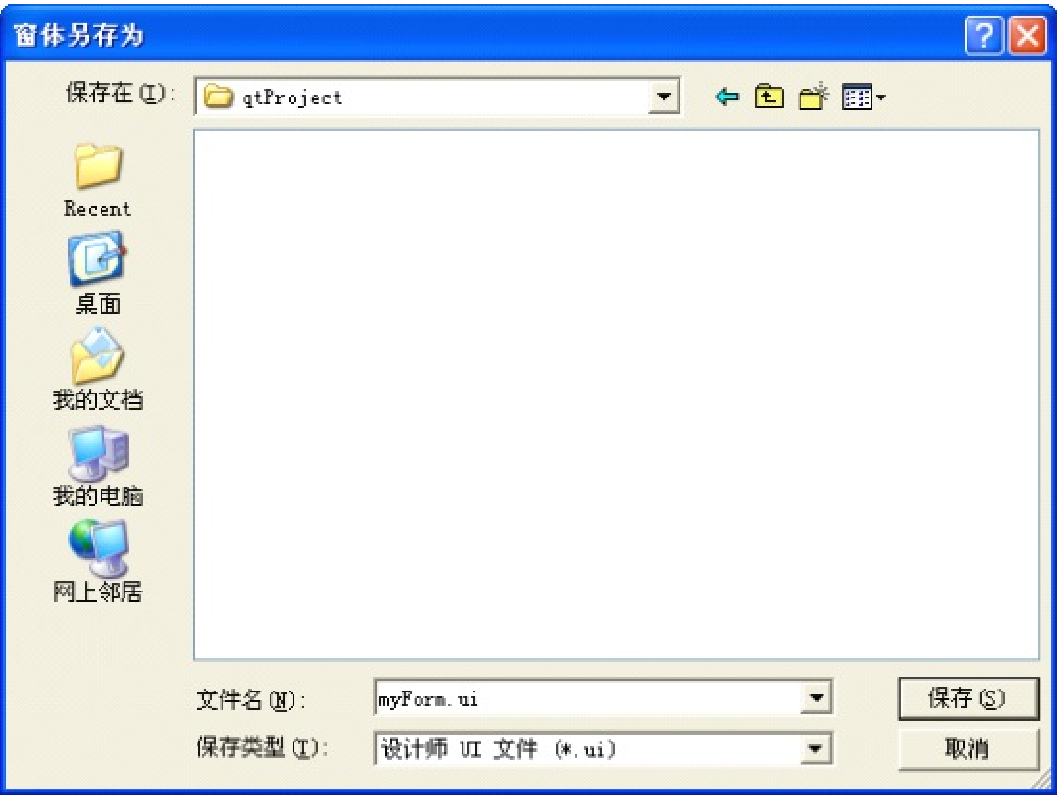


图 5-12 保存窗体

第 5 步，设置界面布局。

与我们习惯的在 MS Visual Studio 中不同，Qt Designer 在设计界面时时，不必 刻意手工调整窗体的大小和精确位置，我们只需选中窗体，并对它们运用布局。例 如，您可以选中一些按钮控件，并通过选择 【水平布局】选项将它们水平并列排放。 采用这种方法，可以使设计更快速，设计完成后，控件会根据最终用户需要的窗体大 小正确缩放。

在 Qt 4.5 中，常见的窗体布局有 6 种，表 5-6 示出了这些布局的种类和功用。

表 5-6 常见的布局类型

布局类型	作用
水平布局	按规则水平排列布局内的窗口部件
垂直布局	按规则垂直排列布局内的窗口部件
分裂器水平布局	按规则水平排列布局内的窗口部件，并将整体作为一个水平分裂器
分裂器垂直布局	按规则垂直排列布局内的窗口部件，并将整体作为一个垂直分裂器
栅格布局	按二维栅格的方式排列布局内的窗口部件
窗体布局中布局	将布局内的窗体部件分成两列，通常用于有输出输出的 GUI 场合

好了，有了上面的指导思想，回到我们的界面设计上来。我们同时按下 Ctrl 和 A键，这就选中了窗体上的所有部件，在它们上面点击鼠标右键，在上下文菜单上依次 选择 【布局】->【栅格布局】，我们就选中栅格布局，这时界面情形如图 5-13 所示，是不是比较美观了。

其实，这样设置还是有一些问题存在，比如最好还要设置窗体的顶级布局（Top Level Layout）等。我们在后面再为大家详细讲解，这里不再深入讨 论了。



图 5-13 采用栅格布局后的效果

第 6 步，设置窗口部件的标签顺序。

说到标签顺序，大家可能听起来有些糊涂，在 Windows 平台上我们通常称作焦点顺序 或者 Tab 顺序，顾名思义，就是按下 Tab 键时，窗口焦点在这些部件间的移动顺序。

设置方法是点击工具栏上的那个带有数字图标的按钮，或者依次点击主菜单的 【编 辑】->【编辑 Tab 顺序】，这就进入了标签设置模式，如图 5-14 所示，窗体中各个具有获 得焦点能力的部件上会出现一个蓝色的小框，框内的数字表示该部件的标签顺序，即焦点顺 序。我们可以通过单击蓝色小框来修改标签顺序，被点中的小框将变为红色，完成设置后按 下 F3 键，切换回到编辑窗口部件模式。

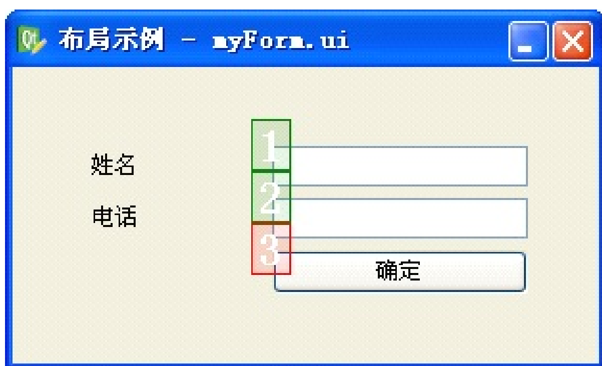


图 5-14 设置窗口部件的标签（Tab）顺序

第 7 步，设置信号与槽

按下 F4 键，或者依次点击【编辑】->【编辑信号/槽】，进入编辑信号/槽模式，如图5-15 所示。



图 5-15 创建信号/槽

这时单击【确定】按钮，然后拖动鼠标，可以发现有一根红色的类似接地线形状的标志线被拖出，松开鼠标，弹出信号 /槽连接配置窗口，注意选中左下角的 【显示从 Qwidget 继承的信号和槽】按钮，界面上将列出所有可以使用的信号和槽， 如图 5-16 所示。

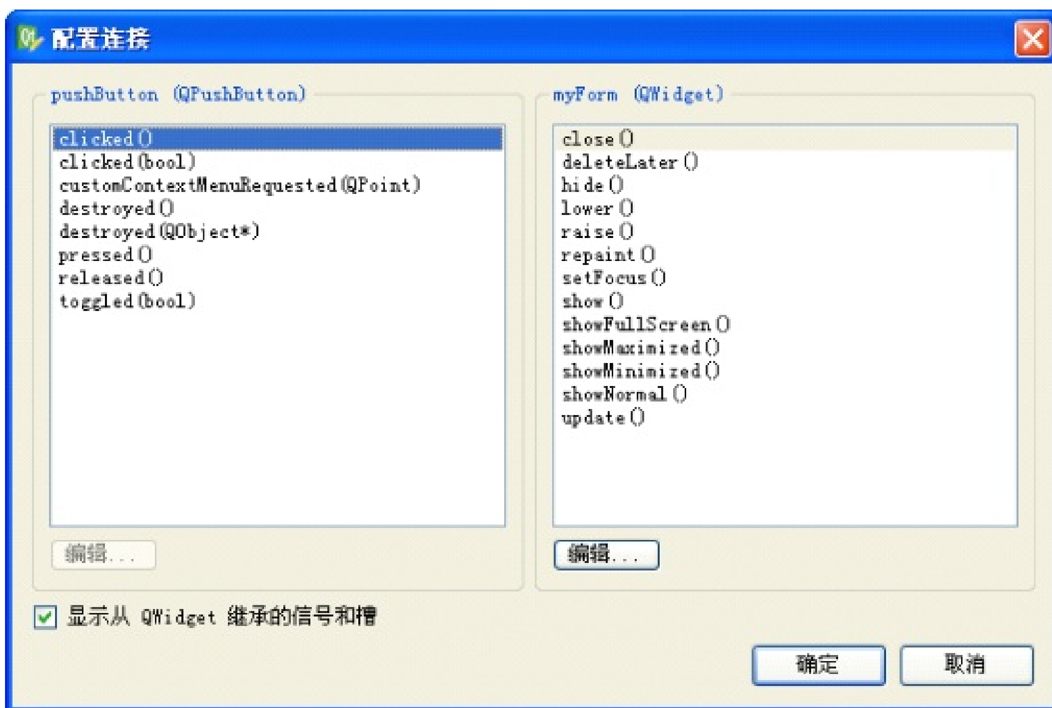


图 5-16 连接信号/槽

在这个连接配置窗口的左侧列出了【确定】按钮的所有信号，右侧列出了 myForm 这个 Widget 的所有槽，选择按钮的 clicked()信号和 myForm 的 close()槽，单击【确定】按 钮，完成设置，如图 5-17 所示。



图 5-17 完成信号/槽的连接

到此，有关 Designer 的操作就结束了，它将生成一个.ui 文件。

注意，Qt Designer 有 4 种编辑模式，即编辑窗口部件模式、编辑信号 /槽模式、编辑 伙伴模式和编辑 Tab 顺序模式，其中编辑伙伴模式不常用到。

关于 Qt Designer 的使用，有非常丰富的内容，这里只是使大家有一个初步的感性认 识，在下面的各章中，笔者将结合具体实例向大家详细的一一介绍。

5.2 使用 Qt Assistant 获取在线文档与帮助

5.2.1 简介

对于大多数复杂的程序来说，在线文档和帮助是必不可少的。 Qt 通过 Qt 助手——一个 帮助文件和文档的在线阅读器，来满足这一需求。 简单的说，Qt Assistant 就是浏览 Qt 参考文档的工具，它具有强大的查询和索引功能，使用时能够比 Web 浏览器更加快速和容 易。而且它可定制，并且可随用户自己的应用程序一起发布，从而形成用户自己的帮助系 统。

开发人员可以将 Qt Assistant 部署为自有应用程序和文档集的帮助浏览器。使用 QAssistantClient 类，可以集成 Qt Assistant。Qt Assistant 使用 QTextEdit 来显示 Qt 的 HTML 参考文档；如有必要，开发人员也可以使用此类来直接实现自己的帮助浏览 器。QTextEdit 支持 HTML 4.0 的子集，它与 Qt 提供的富文本（RichText）类一起使用 时，也可以用来显示其他格式的文档。

5.2.2 Qt 的参考文档

Qt 的参考文档包括大约 3,100 页 HTML 文档，描述 了 Qt 的所有类和工具，并概述 了 Qt 编程的各个方面，所以对于任何一名 Qt 开发人员来说，它都是一个基本工具。 通常 任何一本 Qt 书籍都不能完全覆盖到 Qt 中所有的类和函数，同时也无法提供全部的细节。 所以如果想尽可能的从 Qt 获益，那么就应当尽可能的达到对 Qt 参考文档了如指掌的程 度。

在 Qt 的 doc/html 目录下可以找到 HTML 格式的参考文档，并且可以使用任何一种 Web 浏览器来阅读它。也可以使用 Qt 的帮助浏览器 Qt Assistant，它具有强大的查询和索引功 能，使用时能够比 Web 浏览器更加快速和容易。

另外，可以从 http://doc.trolltech.com 中获取 Qt 的当前版和一些早期版本的在线参考文档。这个网站也选摘了 Qt 季刊（Qt Quarterly）中的一些文章。Qt 季刊是 Qt 程序 员的时事通讯，会发给所有获得 Qt 商业许可协议的人员。

5.2.3 使用 Qt Assistant 1.运行 Qt Assistant

要运行 Qt Assistant，在 Windows 下，如果是采用 Qt SDK 方式安装的 Qt 库，那么你 在程序组里面是找不到 Qt Assistant 的快捷方式的，它被整合到了 Qt Creator 里面，你 可以在运行 Qt Creator 时，按下 F1 键，即可调出 Qt Assistant，或者可以去 Qt 的安装目 录下面找到 Qt Assistant 的执行文件，双击运行或者干脆为它在桌面上设置一个快捷方 式；如果使用框架方式安装 Qt 库的话，可以在程序组里面找到它的快捷方式，点击后即可 运行；在 X11 下，可在命令行终端中输入 assistant 命令；在 Mac OS X Finder 中，只需 双击 assistant 即可。

注意，上述指的是一般的情况，比如各个 Linux 发行版对 Qt Assistant 的命名也有所 不同，甚至有时路径也不完全一样。下面是个具体例子：

如果你是自己编译源代码安装的 Qt，而且没有配置快捷方式的话，可以启动命令行， 进入你的 Qt 安装目录，以笔者的 Red Flag 6.0 为例，依次键入下列命令。

```
$ cd /usr/bin
$ ./assistant-qt4
```

一般的，Qt Assistant 运行起来的效果如图 5-18 所示

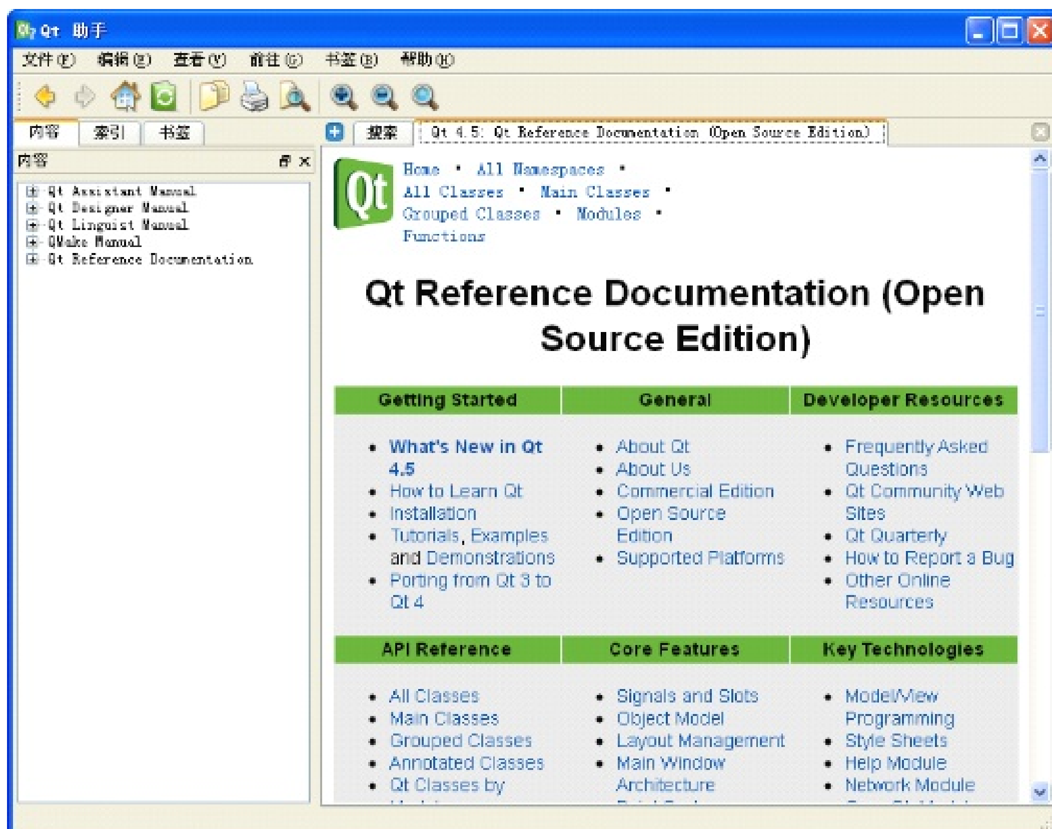


图 5-18 Qt4 Assistant

2.使用方法

其实读者朋友如果用过 Windows 上微软出品的 MSDN 的话，就会发现 Qt4 Assistant 的 界面布局与 MSDN 十分相似，操作方法也是大同小异，很容易上手。

(1) 界面布局

图示出了 Qt Assistant 的界面布局结构，它通常分为 3 个部分，最上面是主窗口，左 侧下面是内容索引窗口，右侧下面是主浏览区。

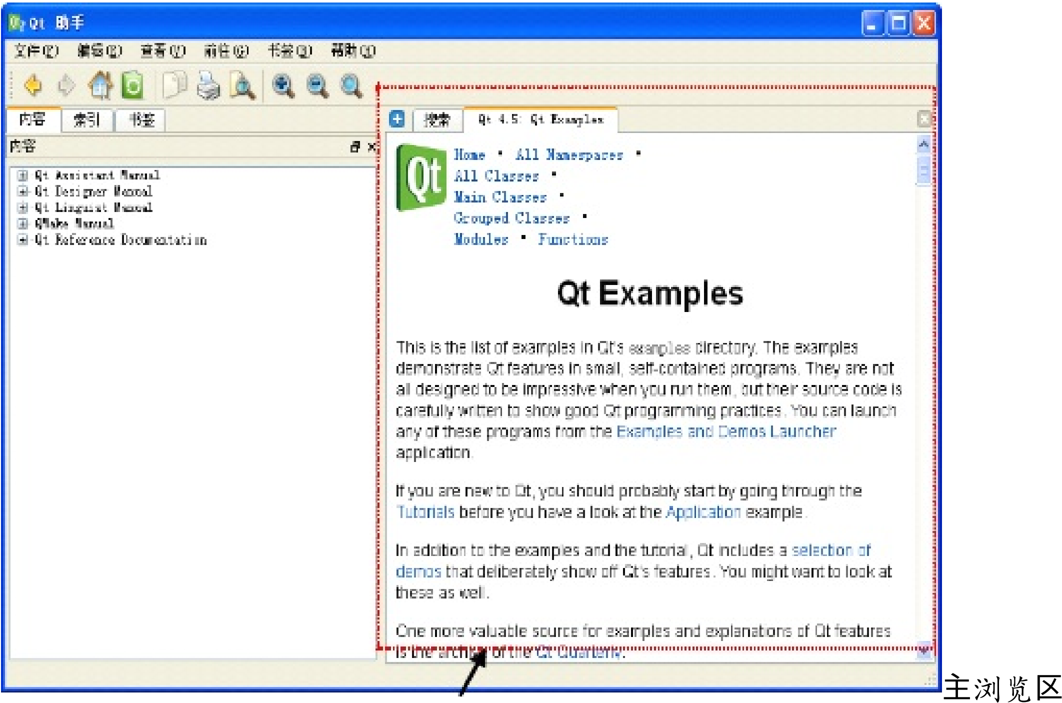


图 5-19 Qt4 Assistant 界面功能分布

(2) 菜单和工具栏

表 5-7 示出了 Qt Assistant 主要菜单项的功用。

表 5-7 Qt Assistant 菜单项

菜单	说明
文件	提供文档设置、打印等相关操作操作命令
编辑	提供首选项设置、复制文本、查找等操作命令
查看	提供工具栏、各个标签页的显隐设置，显示比例设置等操作命令
前往	提供登录 Qt 主页以及在 Qt Assistant 各个页面间导航等操作命令
书签	提供添加书签操作命令
帮助	提供帮助命令

表 5-8 示出了导航工具栏的功用。

表 5-8 Qt Assistant 导航工具栏

名称和图标	功能
	后退
	前进
	登录 Qt 主页
	同步目录

表 5-9 示出了文件和编辑工具栏的功用。

表 5-9 Qt Assistant 文件和编辑工具栏

名称和图标	功能
	复制选中的文本
	打印
	在文本中查找

表 5-10 示出了查看工具栏的功用。

表 5-10 Qt Assistant 查看工具栏

名称和图标	功能
	放大
	缩小
	正常大小

(3) 使用内容索引窗口

默认情况下，内容索引窗口分为 3 个标签页，分别是“内容”、“索引”和“书签”，如图 5-20 所示。其中最常用的是前两个。

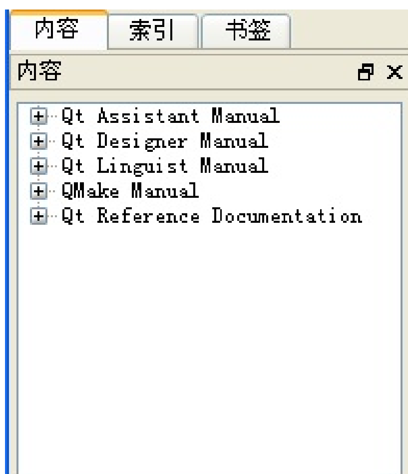


图 5-20 内容索引窗口

内容标签页分成了 5 个大类，分别对应到 Qt Assistant 操作、Qt Designer 操作、Qt Linguist 操作、QMake 使用、Qt 参考文档和例子这几项内容。

如果你能确定你在使用和开发中遇到的问题属于哪种类别的时候，使用 “内容”标签 页是合适和方便的，你只需点击选取某个大类，然后在展开的树形列表中选中某一个具体类 别，双击即可在右面的“主浏览区”内阅读详细的内容，如图 5-21 所示的是你想查看如何 使用 qmake 工具的情形。

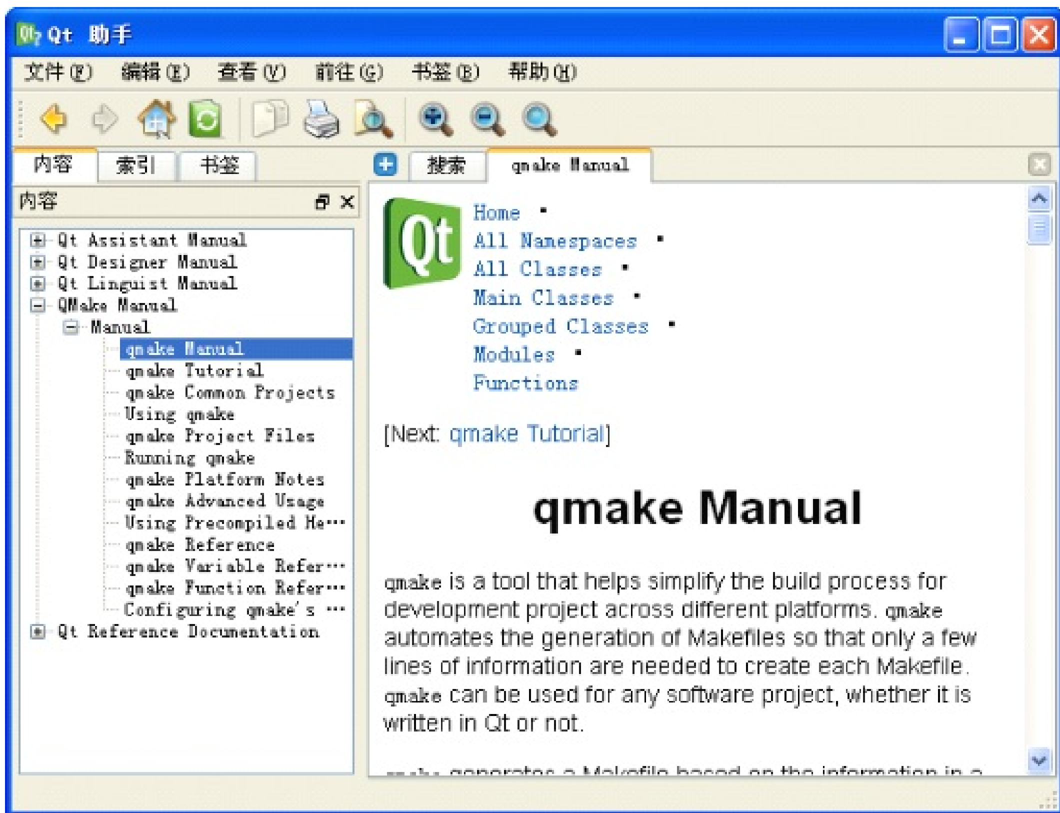


图 5-21 使用内容标签页

可以看到，在主浏览区的内容也是分层次管理起来的，读者可以根据需要进行操作。

(4) 使用“索引”标签页 “索引”标签页是使用频率很高的，因为好多时候我们并不能准确定位出现的问题是属于哪一类的情况，这时候，使用 “索引”标签页，如图 5-22 所示，在“查找”编辑框内输入你的问题或是问题概述或是包含的几个字母都可以， Qt Assistant 将列出所有符合你 的需求的类别，选中你觉得合适的类别，双击它即可在右面的主浏览区内查阅相关内容。

比如我们想浏览 Qt Demo 的情况，在编辑框内输入 qtde，这时 Qt Assistant 已经智能 的显示出了相关的索引结果，然后我们选中 qtdemo，双击它，就可以在右面的主浏览区内 查看了。

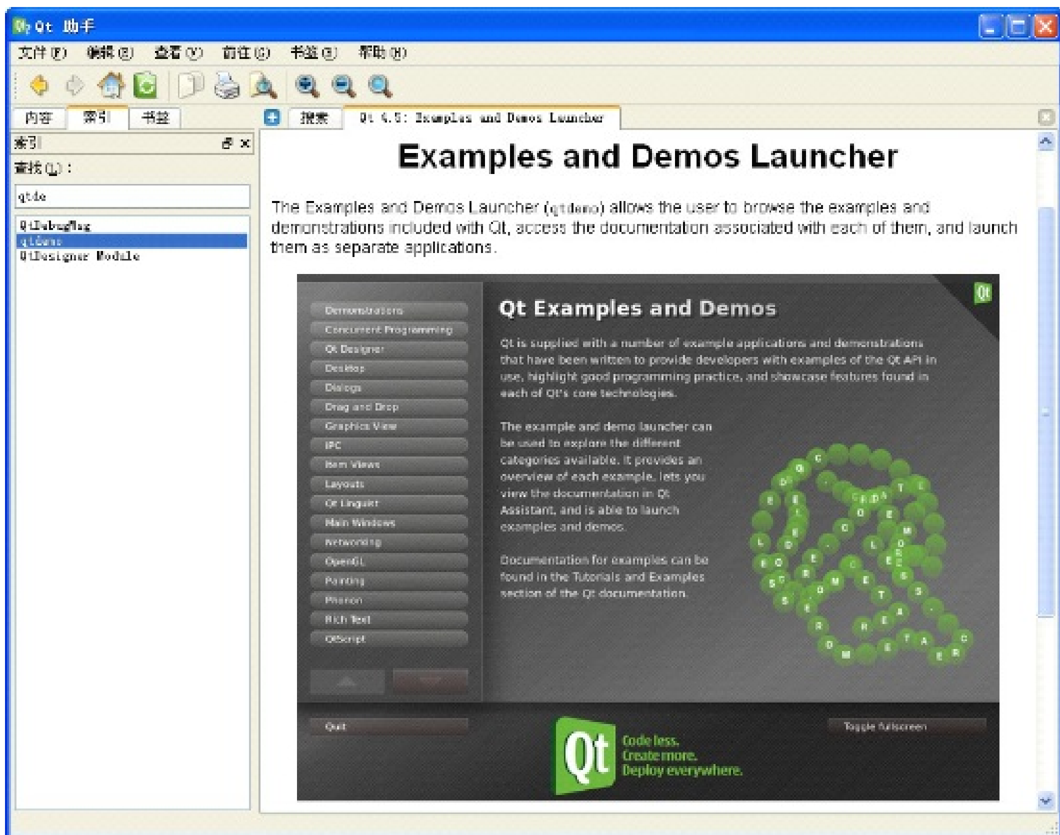


图 5-22 使用索引标签页

(5) 使用“标签”标签页

“标签”标签页也是经常会用到的，当你某次查询时觉得某条资料很有用，希望下次 再来使用，就可以使用“标签”。一个书签页的情形如图 5-23 所示。

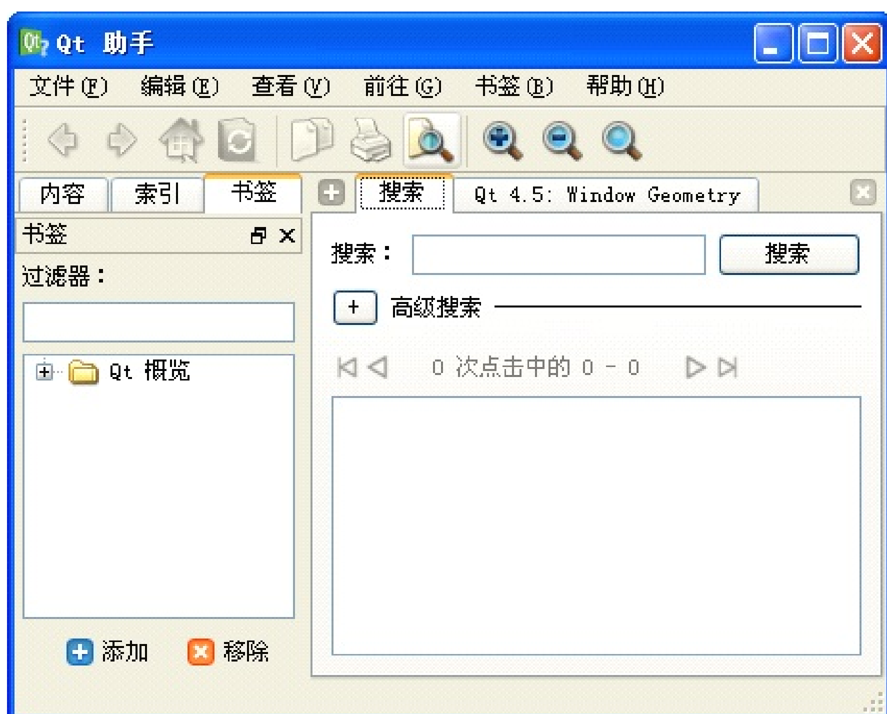


图 5-23 使用“标签”标签页

注意在窗口的左下角有一个蓝色的【添加】按钮和一个红色的【移除】按钮，它们就 是用作添加和删除书签的，你可能还注意到在右边的主浏览区内有一个 【搜索】标签页，下 面就结合这两者的使用说一下如何添加和移除标签 。

第 1 步，在右边的主浏览区内的“搜索”编辑栏内输入你想输入的字符，比如输入 qt，那么主浏览区将会把与 qt 相关的所有条目罗列出来，如图 5-24 所示

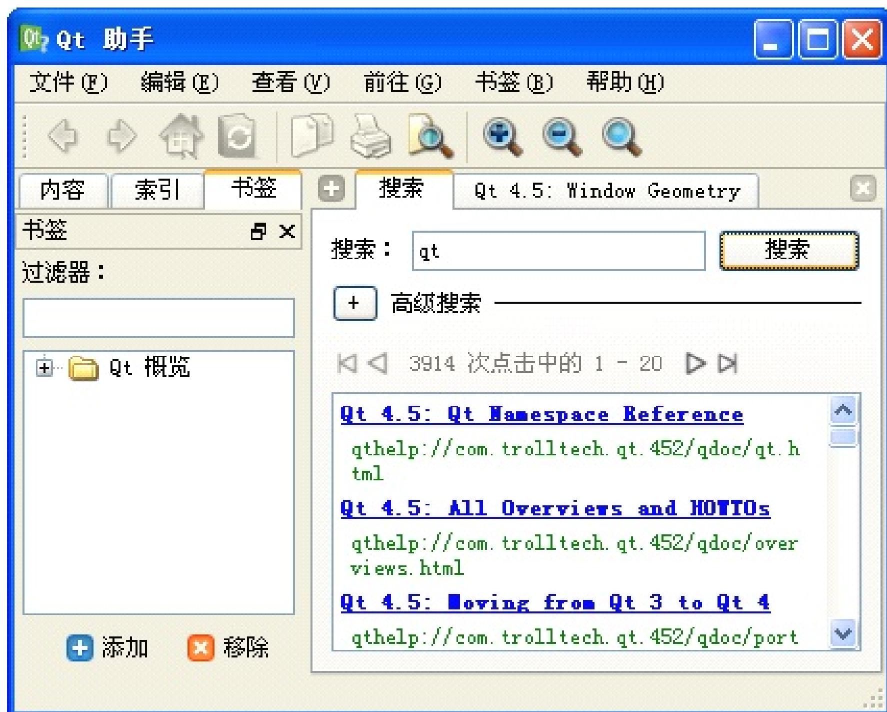


图 5-24 输入关键词进行搜索

第 2 步，你需要选中一个条目（图中蓝色的是条目），比如条目中的第 2 项，如图 5-25 所示，系统将跳转到相应的页面

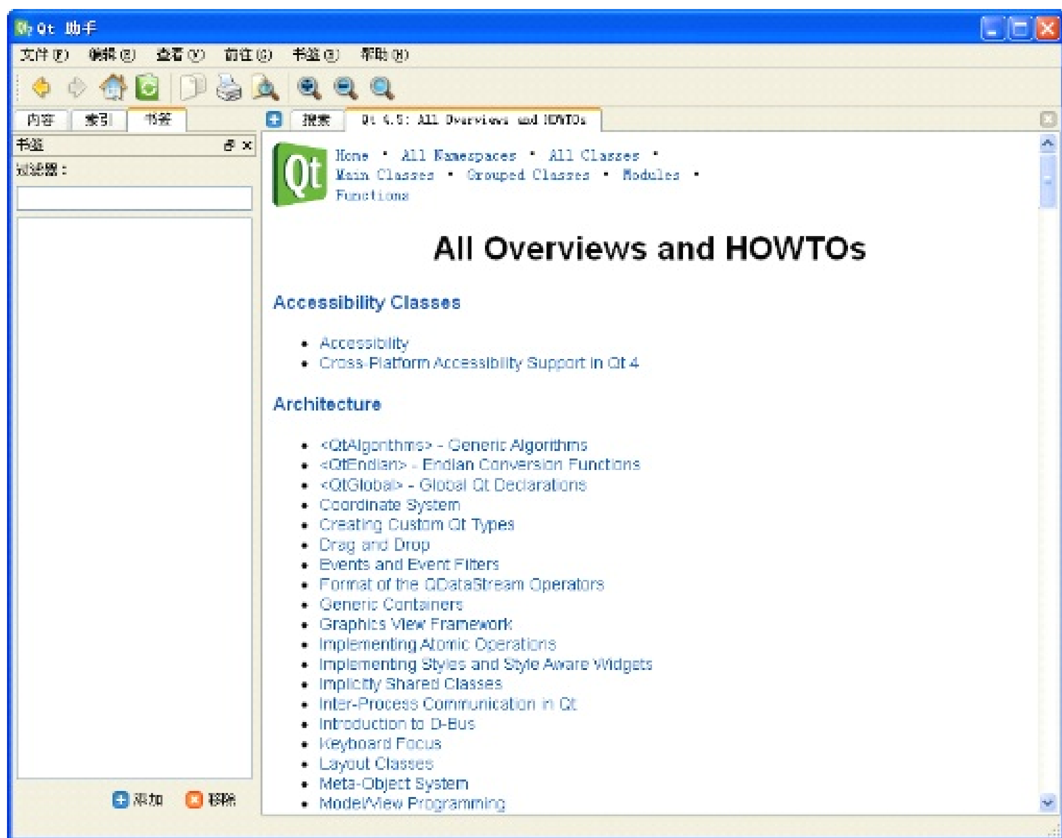


图 5-25 选中一个条目，跳转到相应界面

第 3 步，点击【添加】按钮，Qt Assistant 将弹出“添加书签”对话框，请使用者添 加书签，如图 5-26 所示。

高级配置按钮

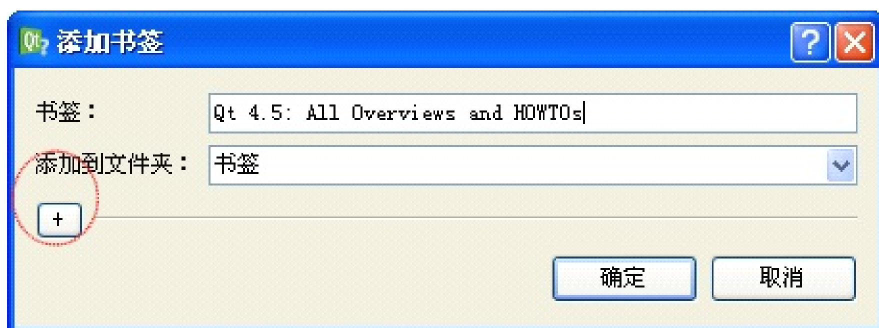


图 5-26 添加书签对话框

如果没有特别要求，就可以在对话框上点击 【确定】按钮，添加书签，如图 5-27 所示。

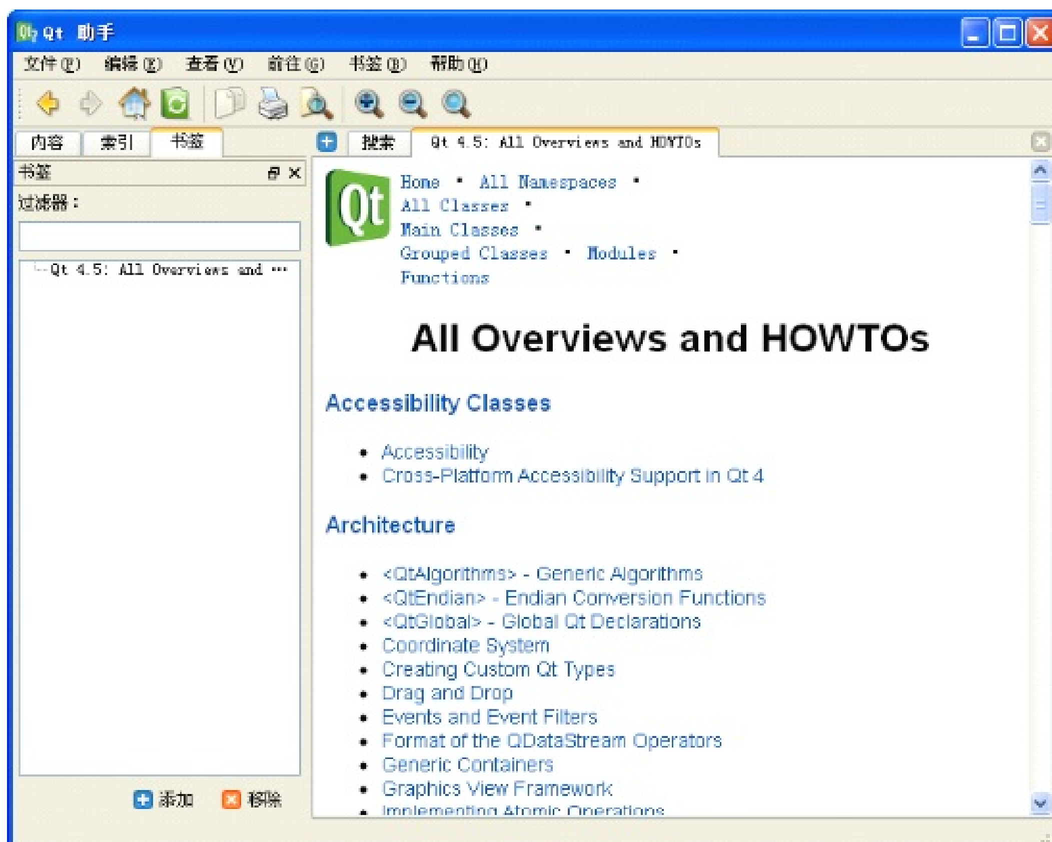


图 5-27 添加书签后的情形

如果还需要进一步定制书签，那就这样做：

第 1 步，点击【添加书签】对话框上点击那个带加号形状的按钮，对话框将伸展开来，显示进一步的配置选项，如图 5-28 所示。



图 5-28 定制书签第 1 步—点击+号按钮

第 2 步，点击【新建文件夹】按钮，这时默认建立了一个名为“新建文件夹”的标签类别，这个过程分别如图 5-29 和图 5-30 所示。



图 5-29 定制书签第 2 步一点击【新建文件夹】按钮

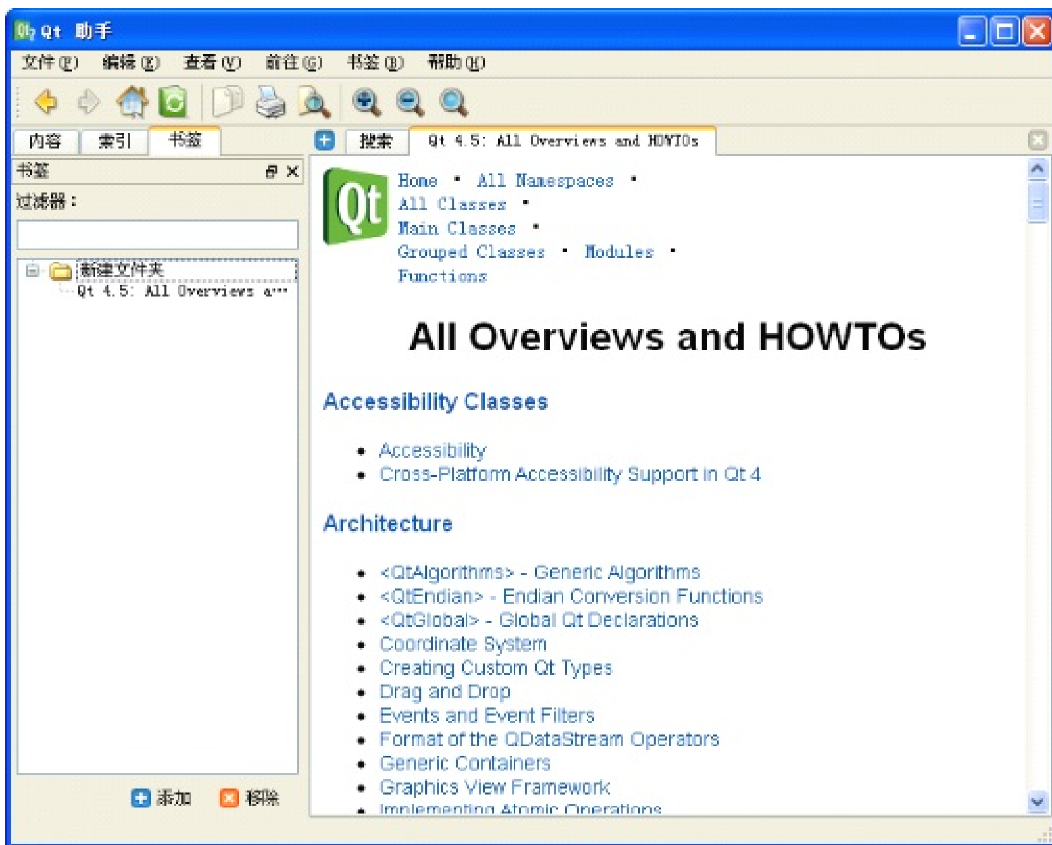


图 5-30 建立好的书签文件夹

第 3 步，在你新建立的文件夹上点击鼠标右键，弹出配置菜单，你可以选择 【删除文 件夹】或是【重命名文件夹】，如图 5-31 所示



图 5-31 第 3 步—配置文件夹

第 4 步，选择【重命名文件夹】后，输入你想定义的文件夹名称，如图 5-32 所示。



图 5-32 重命名文件夹

第 5 步，点击【确定】按钮，Qt Assistant 将为你建立一个标签，如图 5-33 所 示。

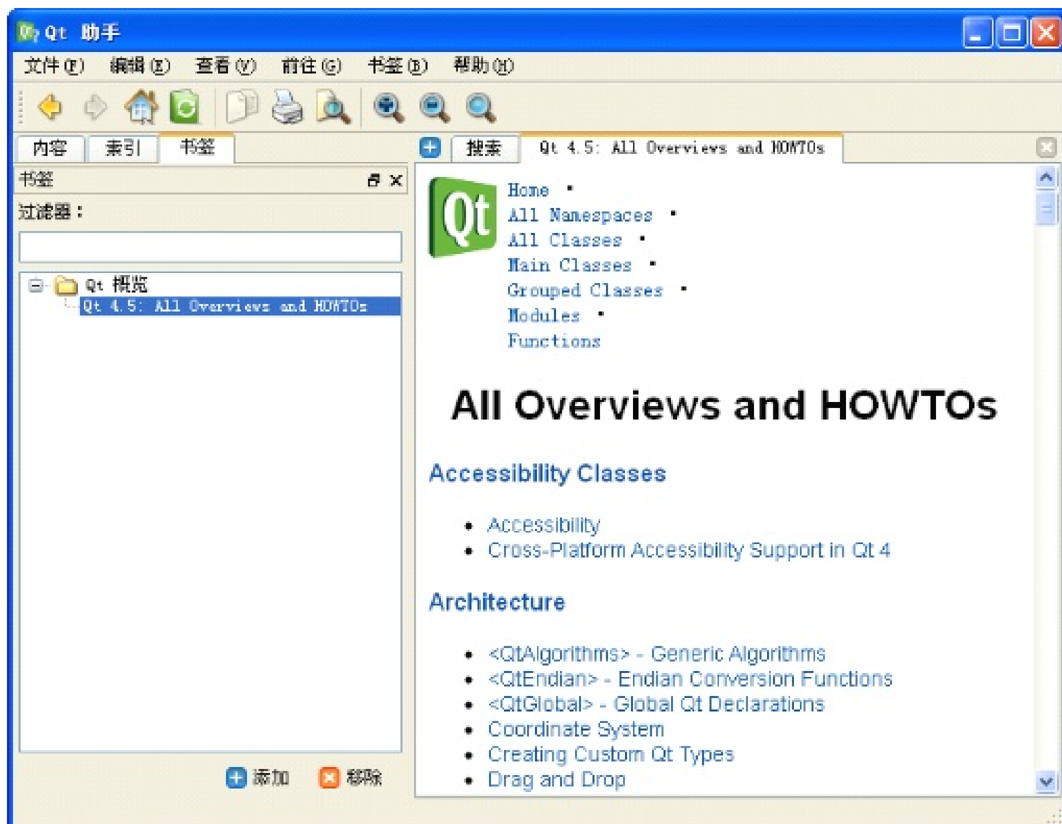


图 5-33 建立好的标签页

第 6 步，依据标签进行检索。在左侧建立好的标签页上双击， Qt Assistant 将跳转到 对应的页面，如图 5-34 所示。

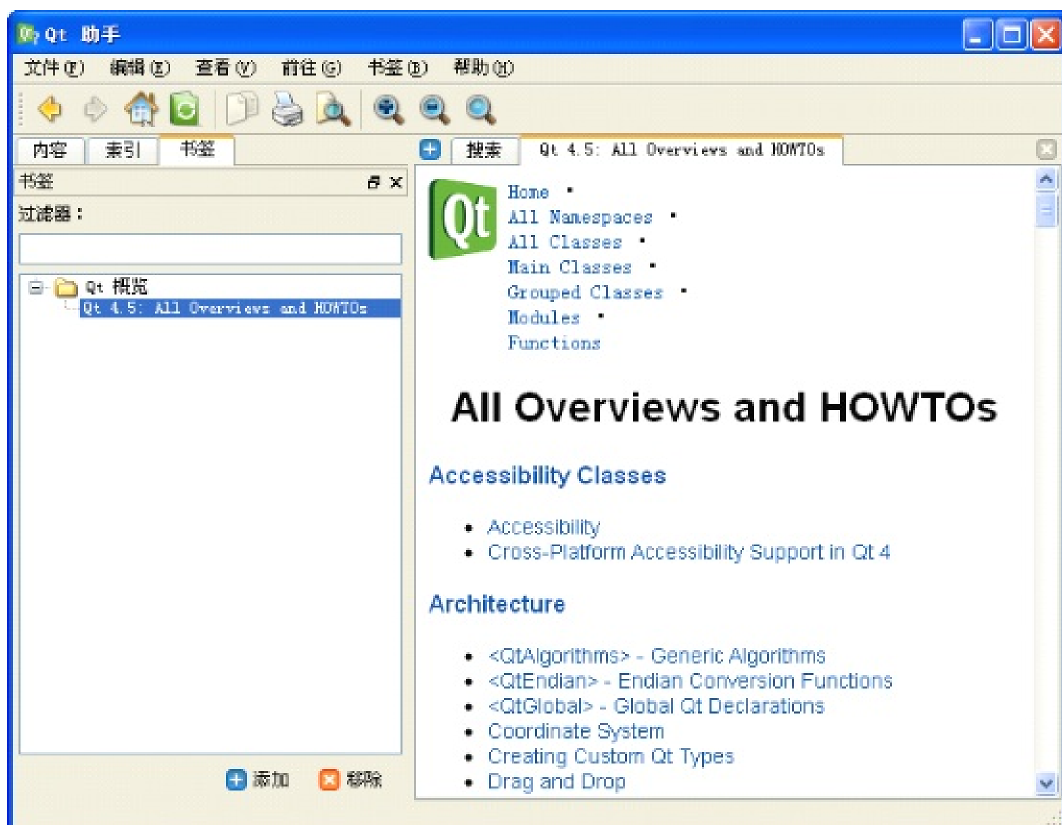


图 5-34 依据标签进行检索

第 7 步，删除标签页。选中建立好的标签，然后点击 【删除】按钮，即可删除标签；也可以选中建立好的文件夹，然后点击 【删除按钮】，系统会出现提示信息，选择【是】按 钮即可删除整个文件夹下的所有标签，如图 5-35 所示。

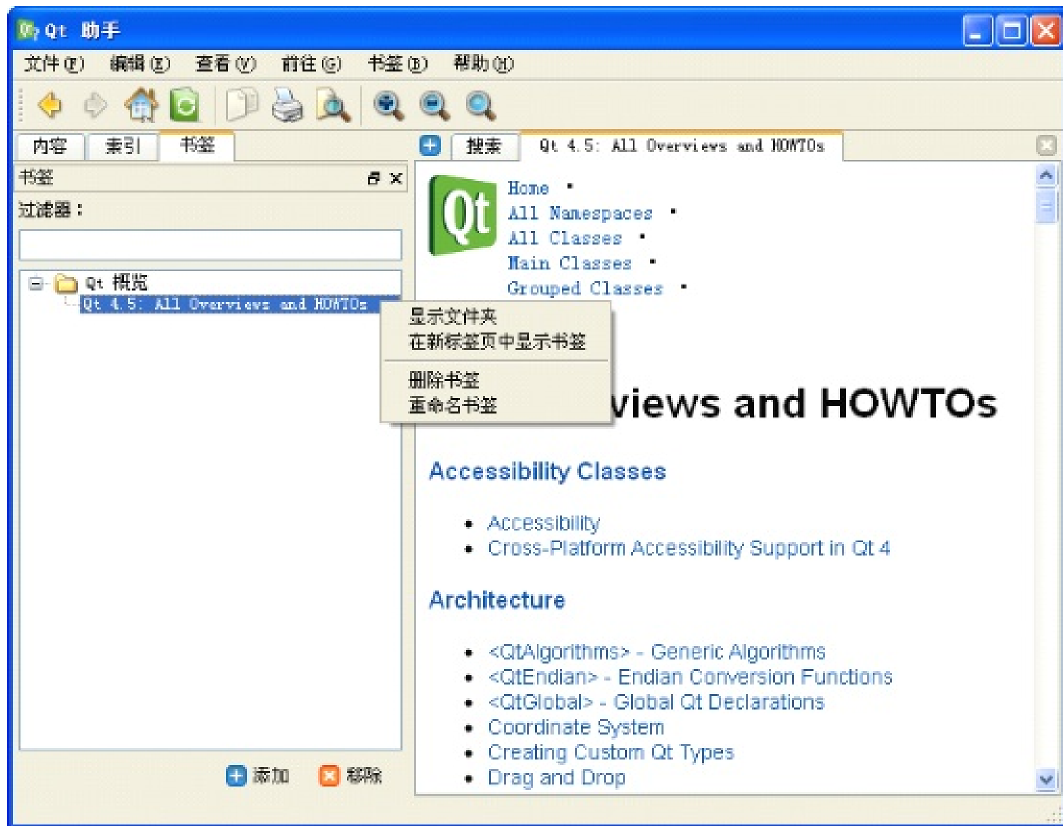


图 5-35 删除文件夹下的书签

至此，关于 Qt Assistant 的使用方法就讲解完了。Qt Assistant 在编程实践中经常会 用到，必须熟练掌握。

5.3 使用 Qt Demo 学习 Qt 应用程序开发

1.简介

Qt Demo 是 Qt 套件之一，它包含了大量的演示和示例程序，基本涵盖了 Qt 编程中的 主要类别，将它与 Qt Assistant 结合使用能够收到很好的效果。

2.运行 Qt Demo

要运行 Qt Demo，在 Windows 下，如果是可依次单击【开始】→【（所有）程序】→【Qt SDK by Nokia v2009.03(OpenSource)】→【Qt Demo】；在 X11 下，可在命令行终端 中输入 assistant 命令；在 Mac OS X Finder 中，只需双击 assistant 即可。

注意，上述指的是一般的情况，请根据你自己安装 Qt 的实际情况进行调整，如采用 SDK 方式安装和采用框架方式安装后，程序组中的指向 Qt Demo 的名字会有所不同，但区别 不大；而各个 Linux 发行版对链接 Qt Demo 的快捷方式命名也有所不同，甚至有时路径也 不完全一样。下面举个具体例子：

如果你是编译源代码来安装的 Qt，并且没有为其配置快捷方式的话，那么可以用命令 行方式运行 Qt Demo，这里以 Red Flag 6.0 为例，进入命令行界面，依次键入:

```
$cd /usr/bin
$./qtdemo-qt4
```

即可运行 Qt Demo 了。

通常 Qt Demo 运行起来的样子如图 5-36 所示，比较遗憾的是目前 Qt4 Demo 还是全英 文的，这就要求使用者对计算机相关的计算机英语有熟练的掌握，如果能够做到这一点，那 么使用 Qt4 Demo 并不是一件困难的事情。

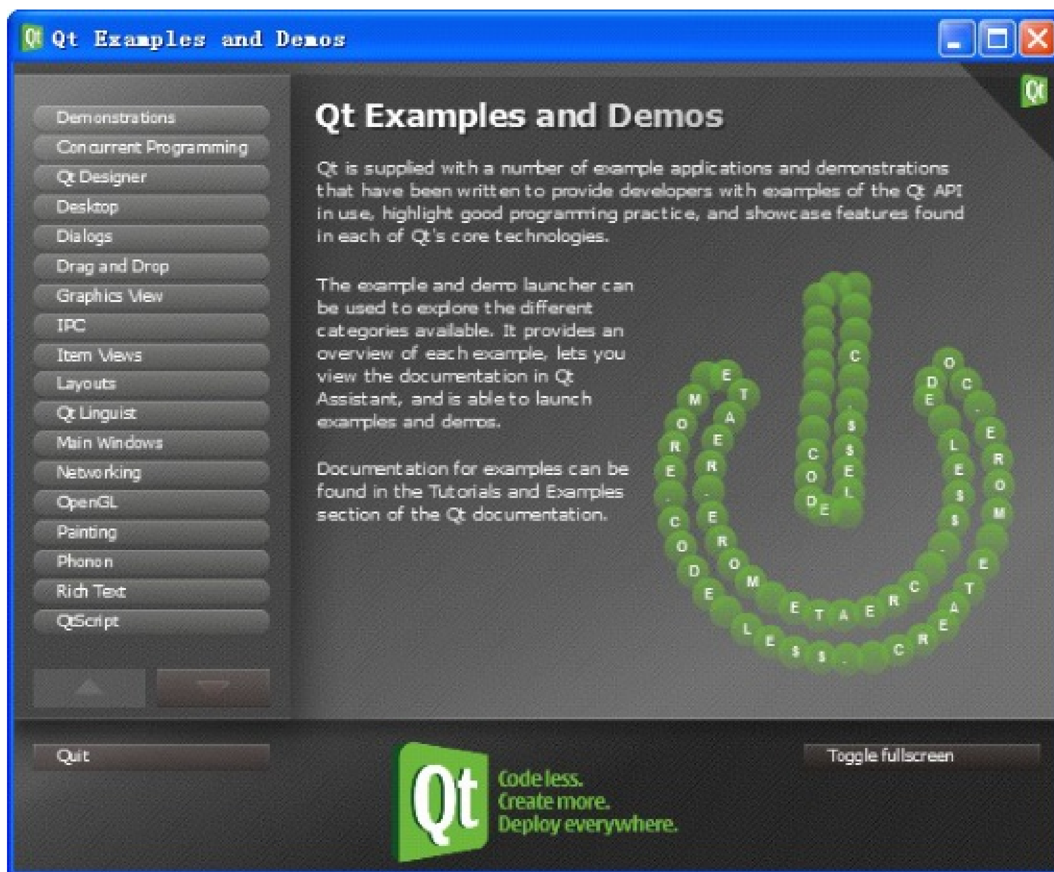


图 5-36 Qt4 Demo 的样子

3.使用 Qt Demo

Qt4 Demo 的界面十分清爽，左边列出了可供参考示例的类别，右边则是这个类别的概 述，在左下角还有导航按钮，可以在各个不同页面间跳转。

在界面的最下方中间是 Qt 的标志，左边是【退出】按钮，用于退出 Qt4 Demo，右边是 用于切换全屏显示和正常显示的按钮。

下面介绍使用使用 Qt Demo 的主要流程。

第 1 步，选择某一个大类。这里是选择【Dialogs】，方法是用鼠标左键在上面单击一下，Qt4 Demo 将跳转到该类别的页面，如图 5-37 所示。

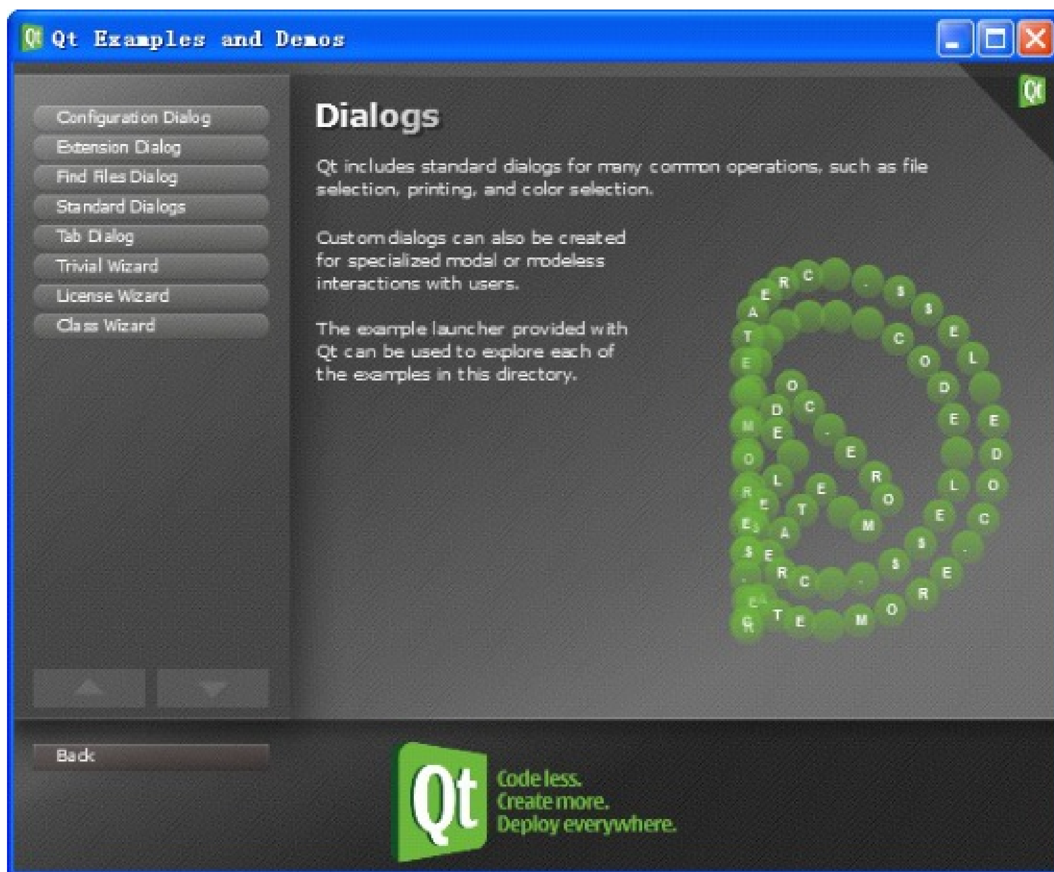


图 5-37 选择 Qt Demo 中大的类别

第 2 步，选择细分类别。在左侧的细分类别中点选一个，如 【Configuration Dialog】，如图 5-38 所示。

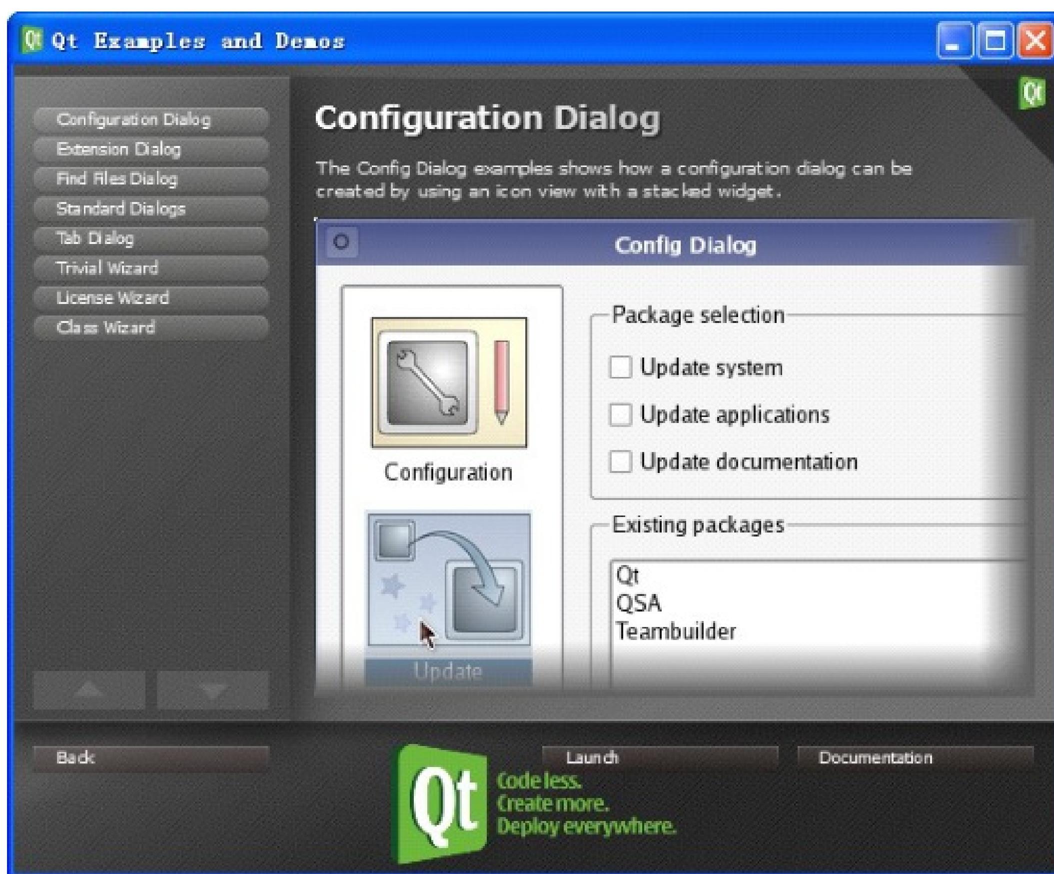


图 5-38 选中【Configuration Dialog】后的情形

第 3 步，查看示例的运行效果。点击【Lauch】按钮，Qt Demo 将运行示例程序，如图 5-39 所示。

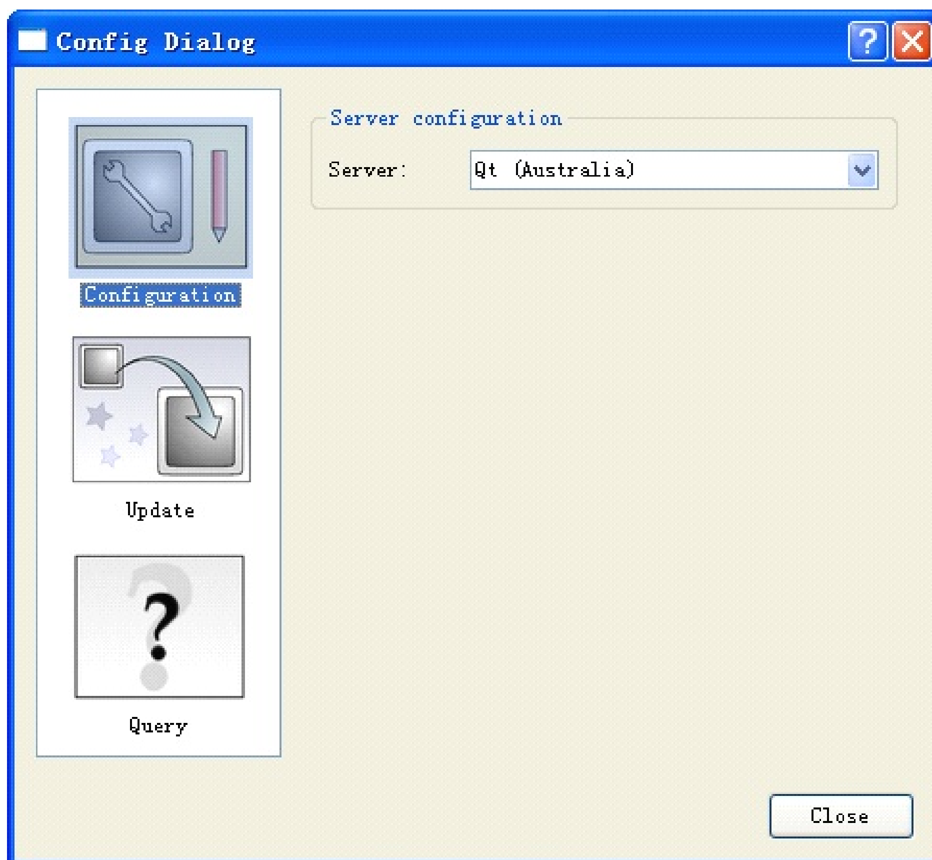


图 5-39 查看示例运行效果

第 4 步，查阅例子代码和参考文档。这实际上是和 Qt Assistant 配合起来使用的，点 击【Documentation】按钮，将调用 Qt Assistant 并切换到相应的页面，如图 5-40 所示。

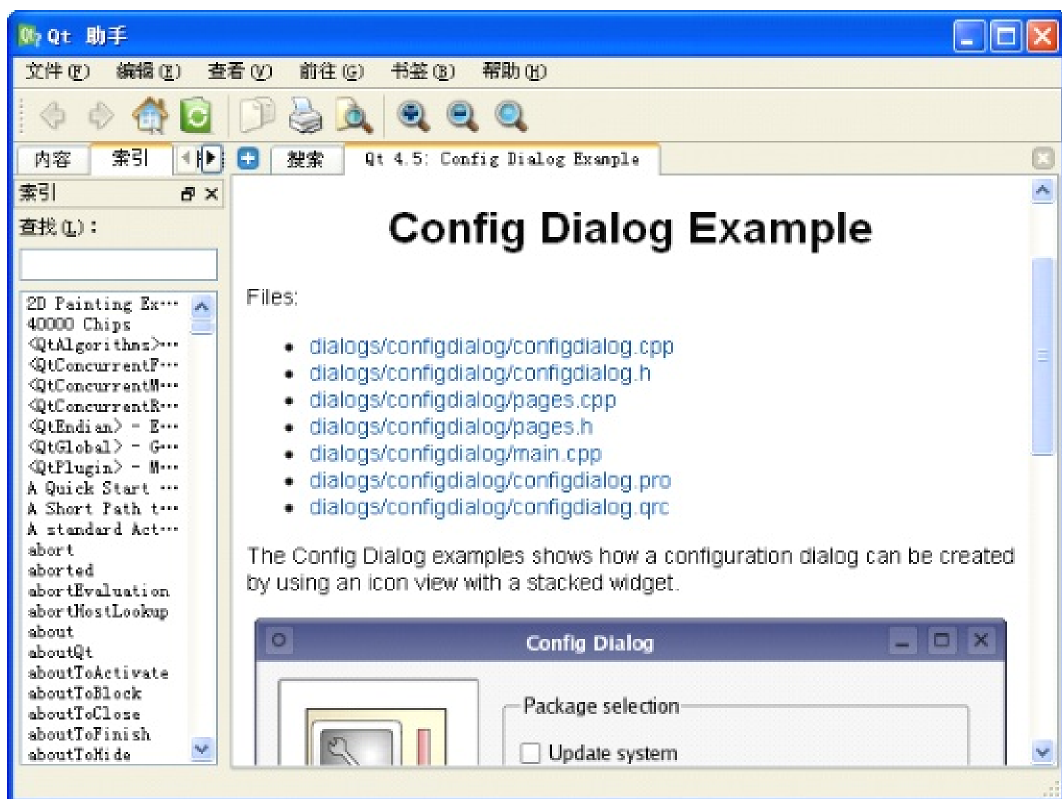


图 5-40 查看例子代码和参考文档

第 5 步，使用代码或文档。下面实际上就是 Qt Assistant 的相关操作了，进入到某一 个源代码文件中，可以浏览、复制源代码，或者稍作修改作为自己代码中的一部分。这里不 再详述，请读者自行实验。

关于 Qt Demo 的使用就就讲解到这里，请读者朋友在使用中与 Qt Assistant 结合起 来，往往能收到事半功倍的效果。

5.4 问题与解答

问：如何使用 Qt Assistant 浏览 Qt 类结构？

答：启动 Qt Assistant 后，点击那个 home 链接标签，就来到它的主页上的“API Reference”小节中的链接提供了浏览 Qt 类的几种不同方式，如图所示。“All Classes” 页面列表会列出 Qt API 的每一个类，而“Main Classes”页面列表只会列出那些最为常用 的类。

需要注意的是，通过继承得到的函数的文档会显示在它的基类中，例如， QPushButton 就没有它自己的 show()函数，因为它是从 QWidget 那里继承的函数，并且没有重载。

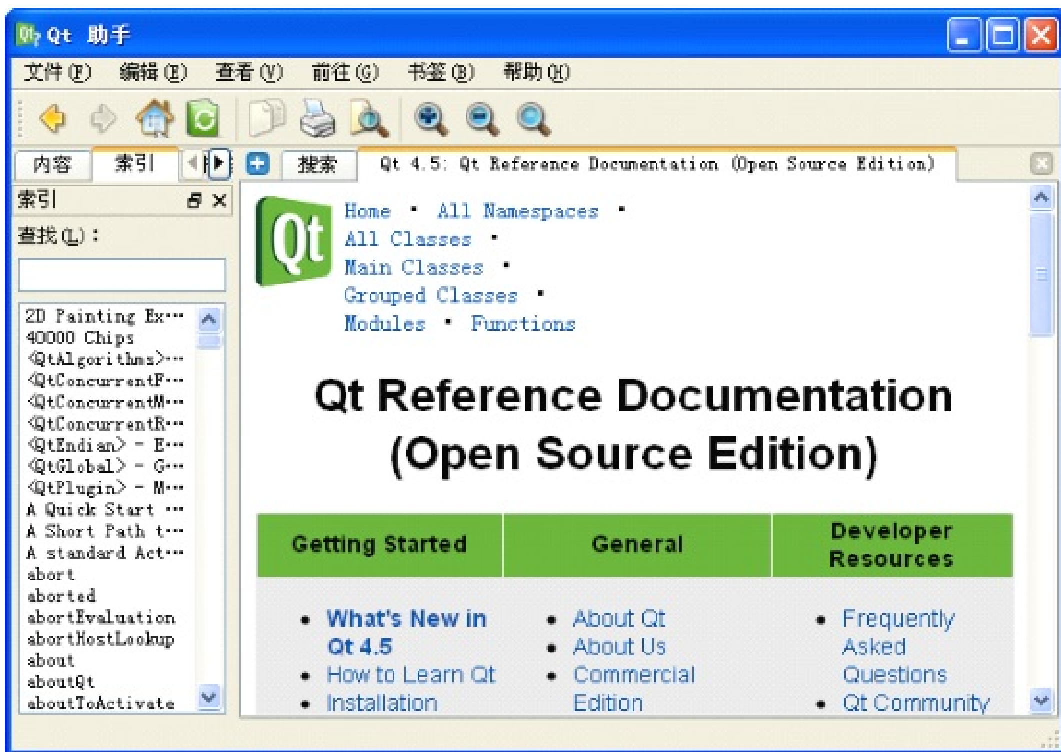


图 5-41 通过主页浏览类结构

问：如果我是使用以前版本的 Qt Designer 设计的用户界面，比如 Qt4.3.4 版，那么 它还可以用在基于 Qt4.5 版构建的工程中吗？

答：这种情况是可以的，Qt Designer 在设计之初就考虑到了这个问题，它是向下兼容 的，但如果使用的是更老的版本，比如 Qt3 甚至 Qt2 制作的界面，就不一定了。因为它们 之间的变化实在很大，尤其是从 Qt3 到 Qt4。Qt 发展到了今天，在桌面平台上，推荐你还 是尽量使用 Qt4；在嵌入式平台则要视你的平台情况选用。

5.5 总结与提高

能够熟练使用 Qt 基本工具是一项必需的技能。本章重点讲述了 Qt Designer、Qt Assistant 以及 Qt Demo 的使用方法和技巧，更多的经验和方法还需要读者朋友在实践中不 断的摸索和总结。

在 Qt 应用程序开发过程中，通常并不是仅仅使用某个单一的工具，而是经常需要综合 使用它们。在下一章里，我们将讲解 Qt 应用程序开发的基本方法和流程，其中的内容也会 涉及到 Qt 工具的综合使用，请大家注意结合阅读。

第 6 章 Qt 4 程序开发方法和流程

本章重点

- 掌握开发 Qt4 程序的基本流程和编译运行方式
- 熟悉 Qt4 工程文件的结构，掌握其书写方法
- 初步了解信号/槽的概念和用法
- 熟练掌握 qmake 的主要用法
- 熟练掌握 3 种主要的 Qt4 应用程序开发方法

从本章开始，我们将真正的步入 Qt4 程序的开发之旅。在经过了前面几章基础知识的 学习后，相信你已经跃跃欲试了吧。在第一节中，我们 来实现一个简单的例子，通过对它的 学习，你会对 Qt 程序开发的基本流程有一个初步的认识；在随后的 内容里面，我将向大家 介绍 Qt 程序的结构，Qt 程序开发的几种不同方法以及它们的适用场合。

本章的例子虽然简单，但它是 Qt 程序开发之旅的第一步，而这在这一章里用到了前面 介绍的许多知识和技能，通过本章也好检验一下学习前几章的效果。

6.1 开发方法

在开发 Qt4 应用程序时，有几种常见的做法。 1.全部采用手写代码，在命令行下完成编译和运行

这种方式是最基础、最基本的，使用它的感觉有点像练武术时的扎马步，一招一式非 常清楚。它最锻炼开发者的技能，因为每一步都不能含糊，开发者需要对编译系统、 Qt 基 础知识有着非常扎实的了解。笔者熟知的许多的 Qt 编程的“老鸟”中，很多人都青睐这种 方式。它的缺点是在一般规模的应用中，还足以胜任，但如果是大型的、多人参与的工程开 发和项目研制，它就有些不方便了，比如如何协同开发、如何进行版本控制管理等等问题都 会变得难以解决。

2.在集成开发环境（IDE）中采用手写代码（包括设计界面），使用 IDE 完成编译和运行

这种方式的好处是可以借助 IDE 来管理工程要素，摒弃了手工的方式，不必太关注工程文件中的一些细节，并且可以借助调试和图形化工具来快速开发；缺点是 IDE 并不是智 能的无可挑剔，它替你完成的一些事情 往往会不如你所愿，很多情况下，你还是需要命令行工具来辅助。

3.使用 Qt Designer 设计界面，使用 IDE 完成编译和运行 这种方式也很常见，开发者使用 Qt Designer 设计界面元素，然后把工程文件的生成、管理，程序的编译运行都交给 IDE 来处理。这种方式的好处可以方便快速的对界面进行修改，在界面元素需要经常变动的情况下，效率比较高；缺点是使用 Qt Designer 生成的代码量比较庞大，由于好多都是自动生成的，阅读代码和调试程序相对比较困难。

对于初学者而言，采用第 3 种方式最容易“入门”,但基础可能打得不太扎实，因为这 些集成式的工具为开发者做了太多的事情，在它们形成的层层布幔之下，隐藏了 Qt 的核心 机制与原理，所以不太容易理解和掌握 Qt 编程的本质。所以呢，我向初学 Qt 的读者朋友 推荐第 1 种方法，先一点一点的做起，待掌握了基础技能后，再快马加鞭也不迟。

6.2 Hello Qt

在这一节中，将通过一个“Hello Qt！”程序，向大家介绍 Qt4 程序编译运行的流 程，并且将采用上面所述的 3 种方法分别介绍，使大家能够对比学习。为什么选择这个 “老套”的程序呢，大家可以看到它虽然简单，却是“麻雀虽小，五脏俱全”。基本上每个 讲述编程的书籍开头不是“Hello World”就是“Welcome You”，好多世界级的大师都这么 做，肯定是有道理的，并且这已经是讲述编程的书籍的“约定俗成”了。

6.2.1 基本流程

1. 成功安装 Qt 4.5

2. 正确配置环境变量（Windows 环境通常不需要）

3. 书写源代码

将下面的源代码保存至一个名为 helloqt.cpp 的文件，并把它放进一个名为 helloqt 的目录中，你可以使用任何一种文本编辑器完成此项工作，比如 Windows 平台的记事本， Linux 下的 vim，Mac OS X 下的 TextEdit。

4. 生成项目文件

在命令行下，进入 helloqt 目录，输入如下命令，生成一个与平台无关的项目文件 helloqt.pro：

```
qmake –project
```

5.生成 makefile 文件

然后输入如下命令，从这个项目文件生成一个与平台相关的 makefile 文件：

```
qmake helloqt.pro
```

6.运行 make 构建程序

在 X11 以及 Mac 下，键入 make；

在 Windows 上，如果使用的是开源版的 Qt，键入 mingw32-make.exe；如果使用的是商 业版的 Qt，则输入 nmake；

7. 运行程序

在 Windows 下，输入 helloqt，并回车运行； 在 X11 下， 输入./helloqt；

在 Mac OS X 下，输入 open helloqt.app

8.结束程序

要结束该程序，可以直接单击窗口标题上的关闭按钮。

小贴士：这就是 Qt4 程序编译运行的大致顺序，上面的这个流程主要是采用命令行方式进 行的。如果你使用的是 Qt 的商业版和 Microsoft Visual C++作为开发环境，则需要使用 nmake 命令替代 make 命令。还可以通过.pro 文件创建一个 Visual C++工程文件，方法如 下：

```
qmake -project
qmake -tp vc helloqt.pro
```

然后，就可以使用 Visual C++打开生成的工程文件，继续完成编译运行的工作。

如果是在 Mac OS X 的 Xcode 里面使用 Qt，则可以打开命令行，输入如下命令来生成一 个 Xcode 工程文件，继而完成后面的工作：

```
qmake -spec macx-xcode helloqt.pro
```

最后这个程序的运行起来就像图 6-1 所示的那样。



图 6-1 Hello Qt!运行效果

6.2.2 源码分析

本实例实现一个“Hello Qt !”的例子，它是基于对话框的程序，界面上有一个按 钮，上面的字符是“Hello Qt!”，单击该按钮，对话框关闭，程序退出。实例效果如图所 示：

实现代码如下：

```
#include &lt;QApplication&gt;;
#include &lt;QPushButton&gt;;
int main(int argc, char *argv[])
{
    QApplication app(argc,argv);
    QPushButton pushButton( QObject::tr("Hello Qt !") );
    pushButton.show();
    QObject::connect(&pushButton,SIGNAL(clicked()),&app,SLOT(quit()));
    return app.exec();
}
```

第 1 行包含了 QApplication 的定义。Qt 的头文件中最为重要的两个是<QApplication>

和<QCoreApplication>。在本书后面的章节中，你会经常看到它们。所有 Qt GUI 应用程序都需要包含<QApplication>这个文件，若使用的是非 GUI 应用程序，则需要包含

```
&lt;QCoreApplication&gt;。
```

小贴士：在 Qt4 中，对于每个 Qt 类，都有一个与该类同名并且采用大写形式的头文件，在这个头文件中包含了对该类的定义。

第 2 行包含了按钮窗口部件的头文件。

小贴士：在 Qt4 中，头文件的包含可以采用类似于<QApplication>和<QPushButton>的形式，也可以使用<qapplication.h>和<qpushbutton.h>的形式。为什么这么说呢，以 Windows 上安装的 Qt4.5 为例，请进入你的安装路径下的 include 目录，我的是 C:\Qt\2009.03\qt\include\QtGui，在这里你可以找到 QApplication 和 qapplication.h， 用记事本打开 QApplication 这个文件，你就会发现里面只有一行代码：#include "qapplication.h"。

第 3 行创建了应用程序的入口，Qt 程序以一个 main()函数作为入口，它有 argc 和 argv 两个参数。

第 5 行创建了一个 QApplication 对象，用来管理整个应用程序所用到的资源。每个 Qt 程序都必须有且只有一个 QApplication 对象。这个 QApplication 构造函数需要 argc 和 argv 作为参数，以支持程序的命令行参数。

第 6 行创建了一个 QPushButton 对象，它是一个窗口部件(widget)，并把它的显示文 本设置为“Hello Qt!”。

小贴士：在 Qt 和 UNIX 术语中，窗口部件（widget）通常是指用户界面中的可视化元素， 比如按钮、滚动条和菜单等都是窗口部件。该词起源于 “window gadget”这个词组，语义 有些类似于 Windows 系统中的“控件”（control）的含义，但又不尽相同。

第 7 行调用按钮对象的 show()方法，将按钮显示出来。Qt4 在创建窗口部件的时候， 通常都是隐藏不显示的，可以调用 show()方法来将它们显示出来。这种做法还有一个好 处，就是我们可以先对窗口部件的属性等进行设置，然后再显示出来，从而防止闪烁现象的 出现。

第 8 行使用了 Qt 的信号/槽机制。这一机制是这样运作的，Qt 的窗口部件可以通过发 射信号（signal）来通知应用程序，某个用户动作已经发生或者是窗口部件的某种状态发生 了变化，应用程序通过一个称为槽（ slot）的函数来做出回应和处理。以我们这个程序为 例，当用户使用鼠标左键单击那个 “Hello Qt!”按钮时，该按钮就会发射一个 clicked() 信号，根据我们的设置，QApplication 对象的 quit()槽负责响应这个信号，它执行退出应 用程序的操作。

这里大家不必深究这个内容，只需要知道“它们就是这么用的”即可。在本节的后面

会讲述信号/槽机制的用法，更为深入的内容，可以阅读第 13 章。

第 9 行调用 QApplication 的 exec()方法，程序进入事件循环，等待用户的动作并适时 做出响应，这里的响应通常就是执行槽函数。 Qt 完成事件处理及显示的工作后，退出应用 程序，并返回 exec()的值。

6.2.3 编译运行

在 6.2.1 节里，我们已经总结了编译运行 Qt 的方法，但恐怕还是有些晦涩难懂，为了 使读者朋友看得更为清晰，现在把它图文解说一下。以 Windows 平台为例，操作系统是 Windows XP SP2 中文版，Qt 是 4.5 SDK，我们从第 4 步开始（前 3 步实在没有什么好说 的，如有不明白的，请回头去看前几章，呵呵）。

依次点击【开始】→【所有程序】→【Qt SDK by Nokia v2009.03(open source)】→【Qt Command Prompt】,启动 Qt 命令行，如图 6-2 所示，每次启动时，命令行会自动为我 们配置好 Qt 所需的环境。

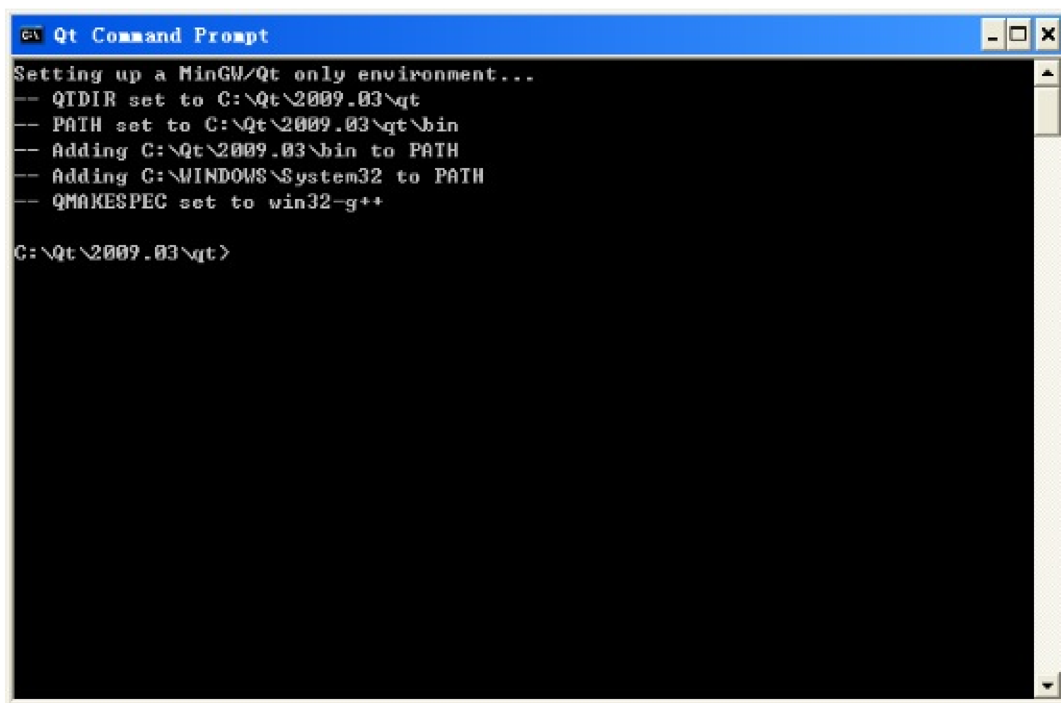


图 6-2 启动 Qt 命令行

接下来切换到你保存的程序文件目录，我这里的情形如图 6-3 所示

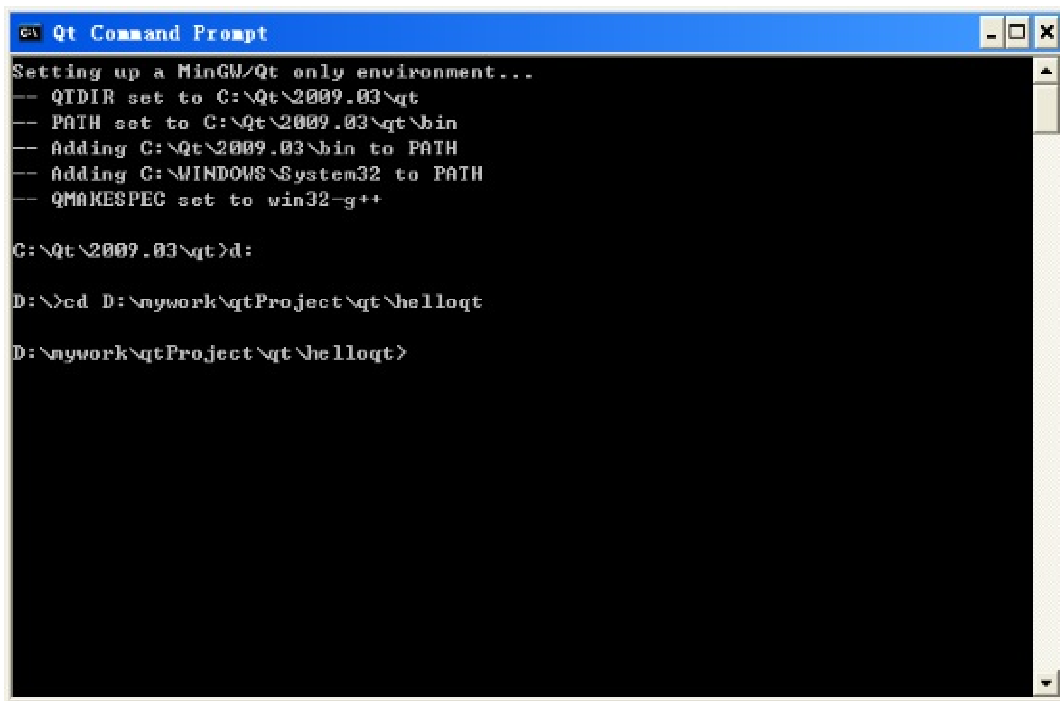


图 6-3 进入程序文件所在的目录

接下来输入 qmake –project，生成项目文件，再输入 dir/p 查看,发现生成的项目文 件名为 helloqt.pro。

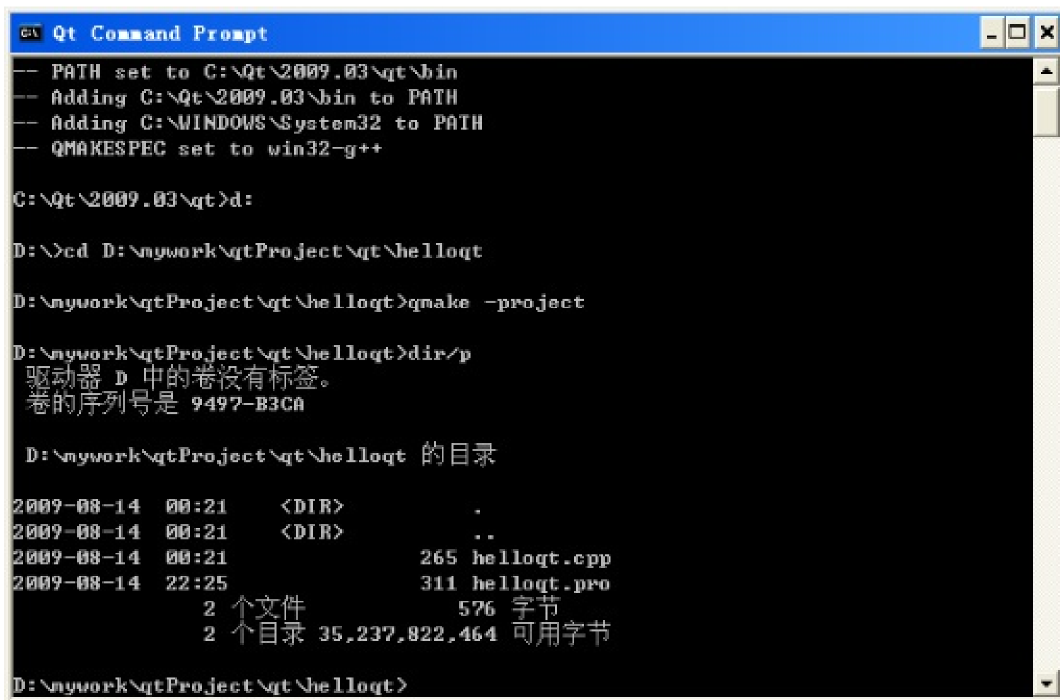


图 6-4 生成项目文件

然后输入 qmake –helloqt.pro，生成 makefile 文件。

这之后再输入 mingw32-make.exe，开始编译程序，情形如图 6-5 所示，编译成功后，将在 debug 目录下生成一个名为 helloqt.exe 的可执行文件。

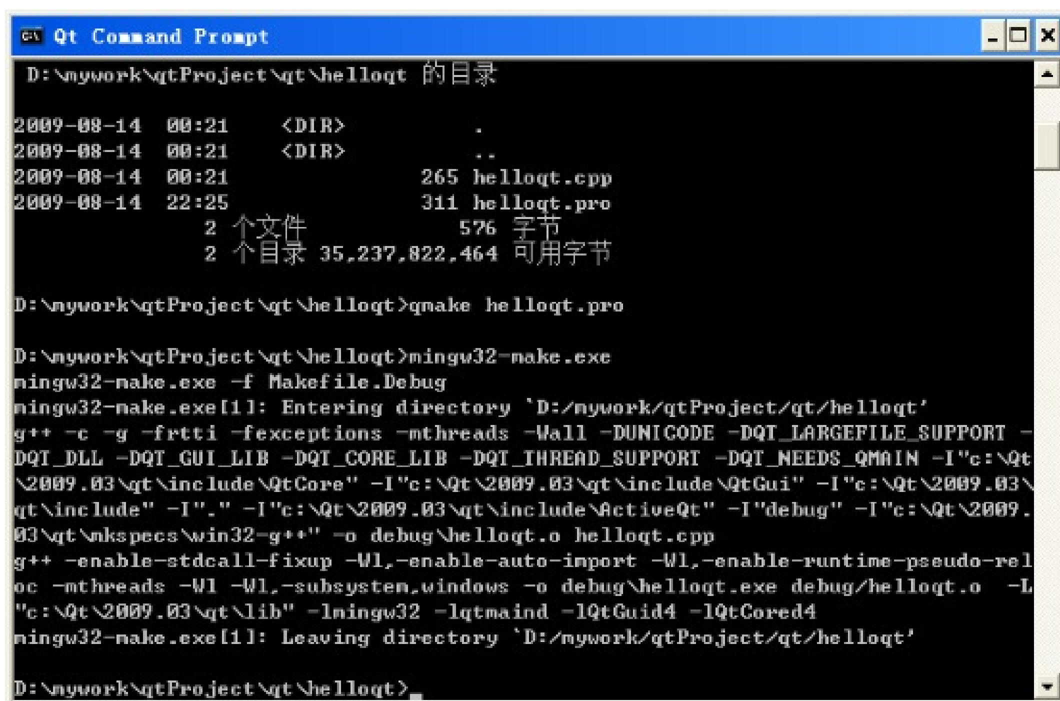


图 6-5 编译程序

进入 debug 目录，输入 helloqt,运行它，程序的样子与图 6-1 所示相同。 好了，到此我们的第一个 Qt4 程序就顺利完成了，它是基于手写代码，并且是采用命令行方式编译运行的。这就是我们所说的第 1 种方法。

6.2.4 第 2 种方法

这里我们以 Qt Creator（关于 Qt Creator 的详细使用方法，请见第 12 章）作为 IDE，流程如下：

1. 成功安装 Qt4.5

2. 正确配置环境变量（Windows 环境通常不需要）

3. 书写源代码

4. 启动 Qt Creator

首先依次点击【开始】→【所有程序】→【Qt SDK by Nokia v2009.03(open source)】→【Qt Creator】,启动 Qt Creator。

然后依次点击主菜单的【Tools】→【Options...】,在弹出的如图所示的对话框中选 中【Qt4】→【Qt Versions】,查看 Qt Creator 识别的 Qt4 的版本情况，如果符合你的实 际情形，就直接点击【OK】按钮退出；如果不符合，你可以对它进行增、删、改，方法比较

简单，根据屏幕的提示完成即可。

注意，这一步很重要，如果 Qt Creator 没有正确识别你的 Qt4 所在的位置，那么接下 来的所有步骤都将变得没有意义。

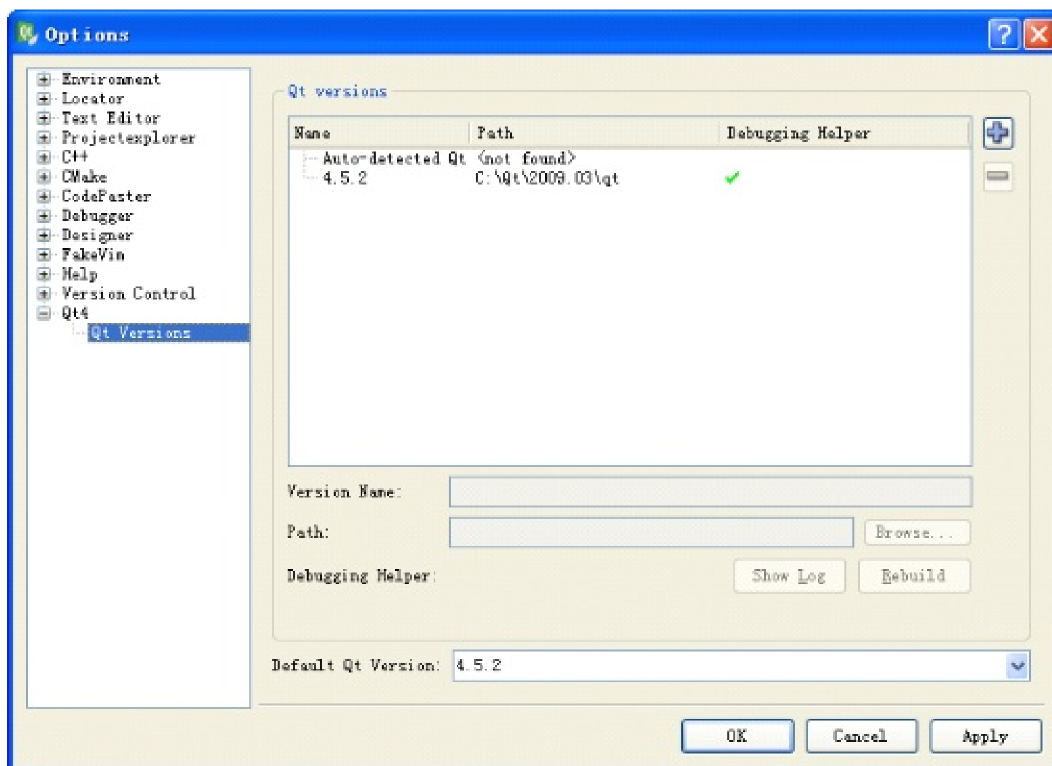


图 6-6 查看并设置 Qt4 环境

5.新建工程

依次点击【File】→【New...】→【Projects】→【Empty Qt4 Project】，如图 6-7 所示，点击【OK】按钮建立一个空的 Qt4 工程。

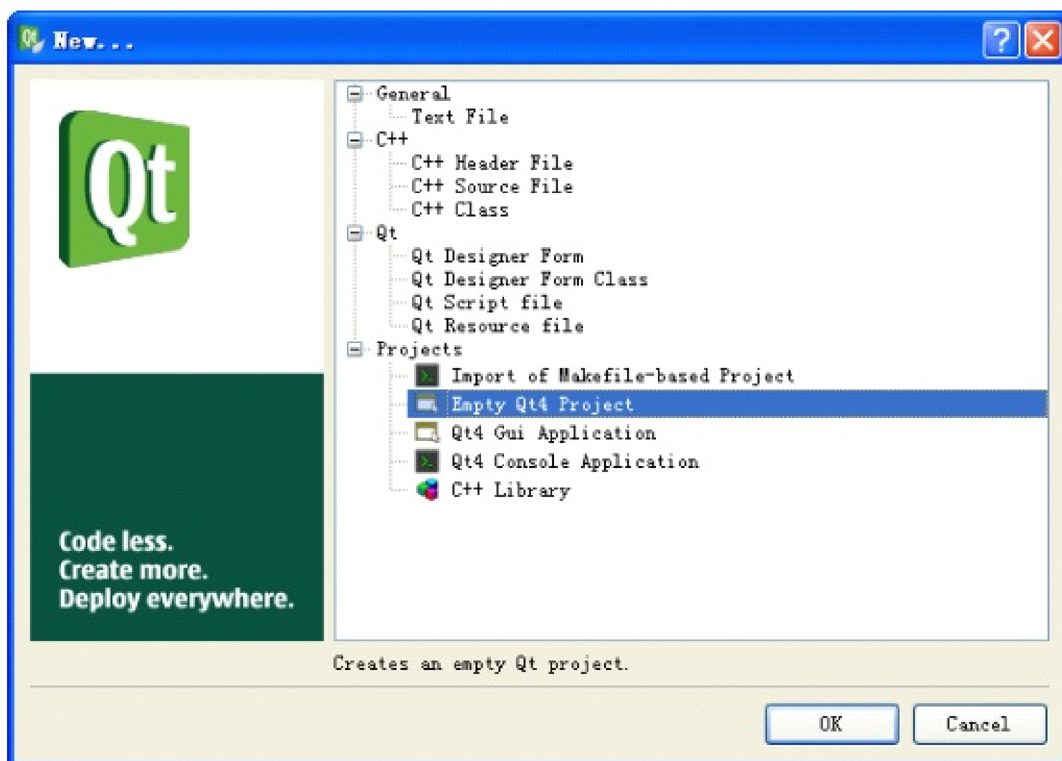


图 6-7 新建 Qt4 工程

接下来弹出的对话框会要求你设置工程的名字和所在的目录，并且会提示你不可采用 特殊的字符，如图 6-8 所示，根据笔者的经验，不要包含空格，最好不要包含中文字符， 切记！

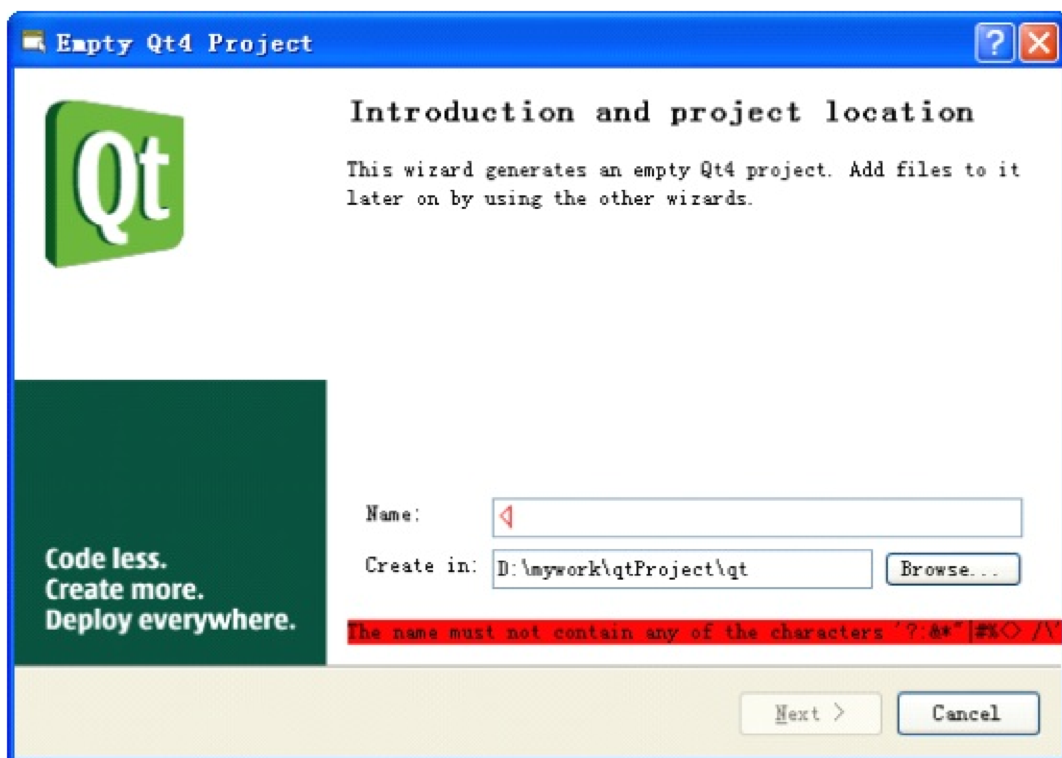


图 6-8 Qt 对于命名的要求

如图 6-9 所示，输入正确的路径和工程名字（这里是 helloqt）后,点击 Next 按钮。

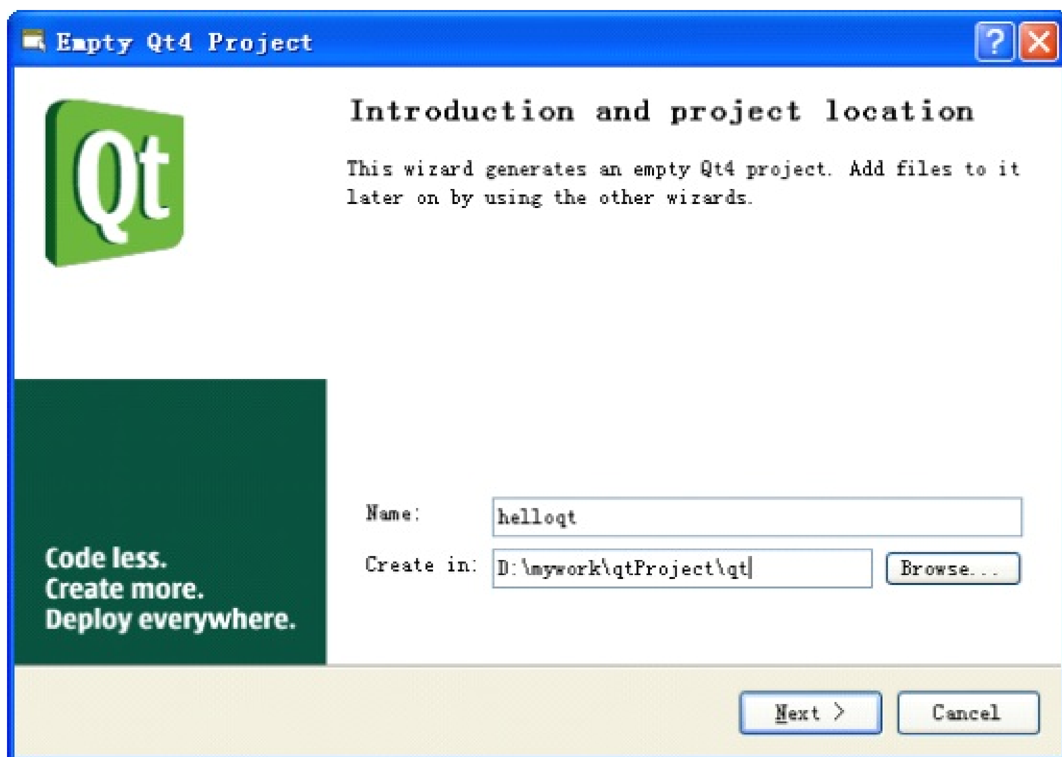


图 6-9 正确的输入工程名称和目录

接下来这一步没有什么好说的，系统向你展示了即将创建的工程的信息，如图 6-10 所示，点击 Finish 按钮，这就生成了一个新的工程，如图 6-11 所示。

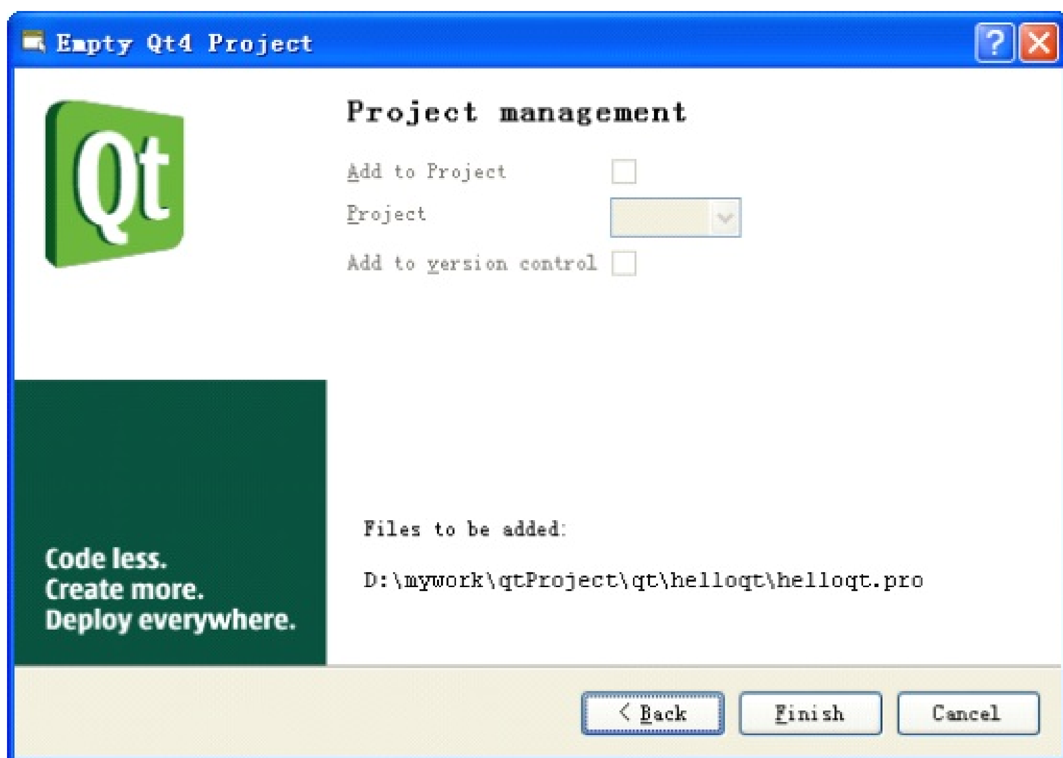


图 6-10 新工程的信息

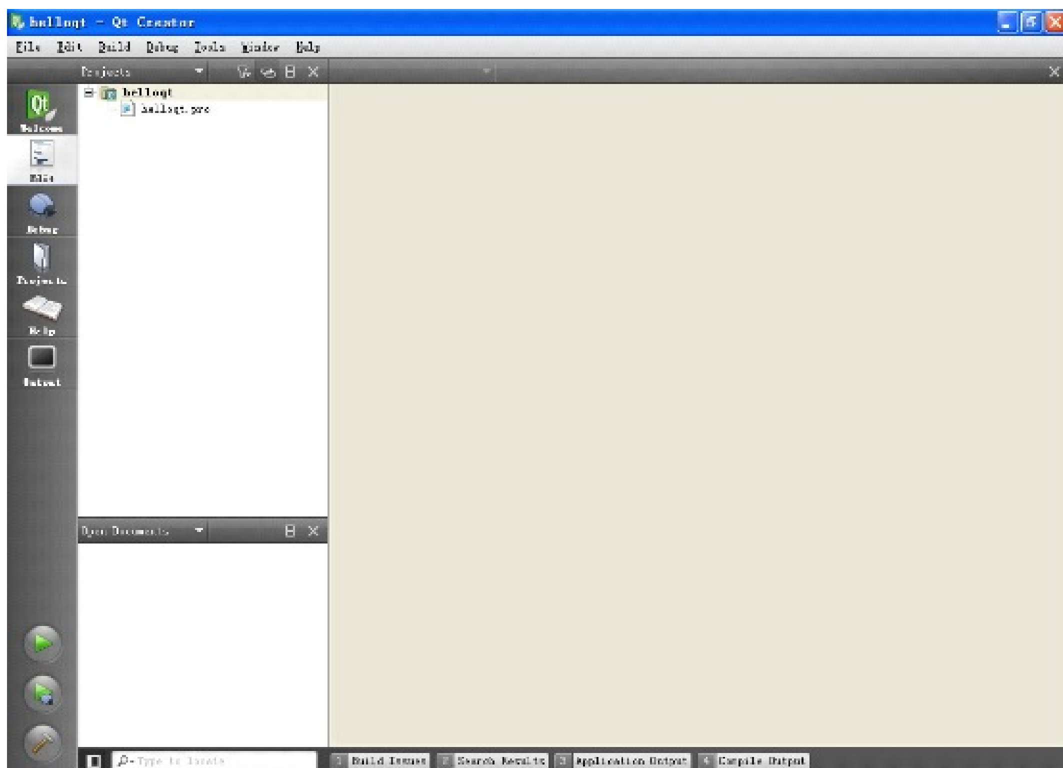


图 6-11 生成了一个新的工程

这时候的工程还是空的，我们需要向里面添加文件。有两种方法，一是添加已经存在的，还有就是新建一个。

下面先说如何添加一个已经存在的文件。 在工程目录上点击鼠标右键，在弹出的上下文菜单上选择 【Add Existing Files...】，如图 6-12 所示。

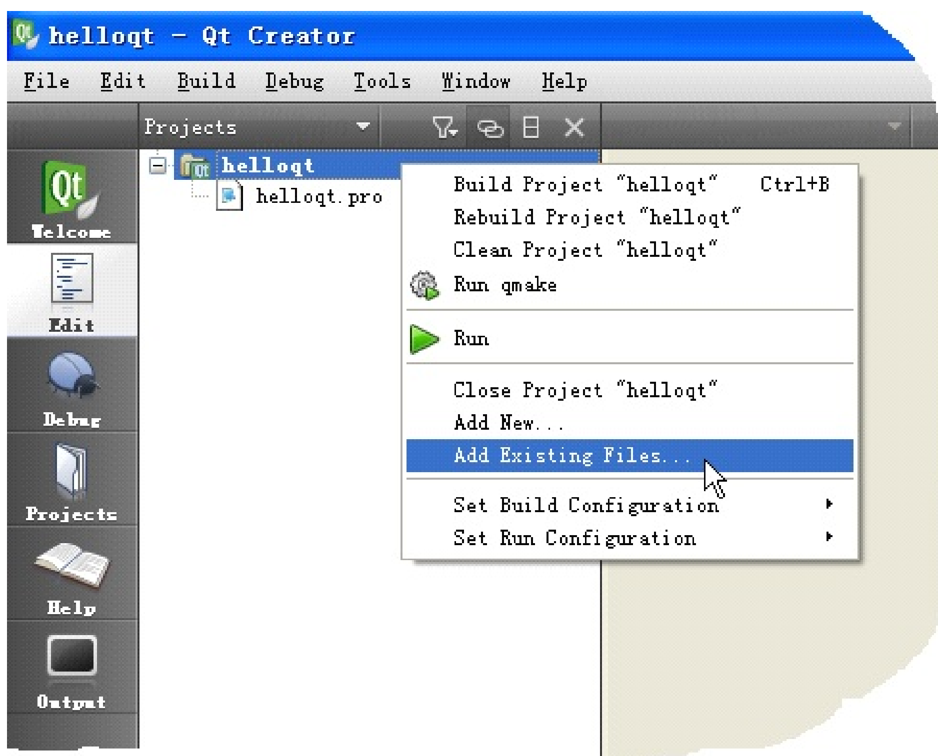


图 6-12 添加已有文件

然后找到你要添加的 helloqt.cpp 的位置，把它加入进去，这时候在工程目录里面就 多了一项你刚才添加的文件，如图 6-13 所示。

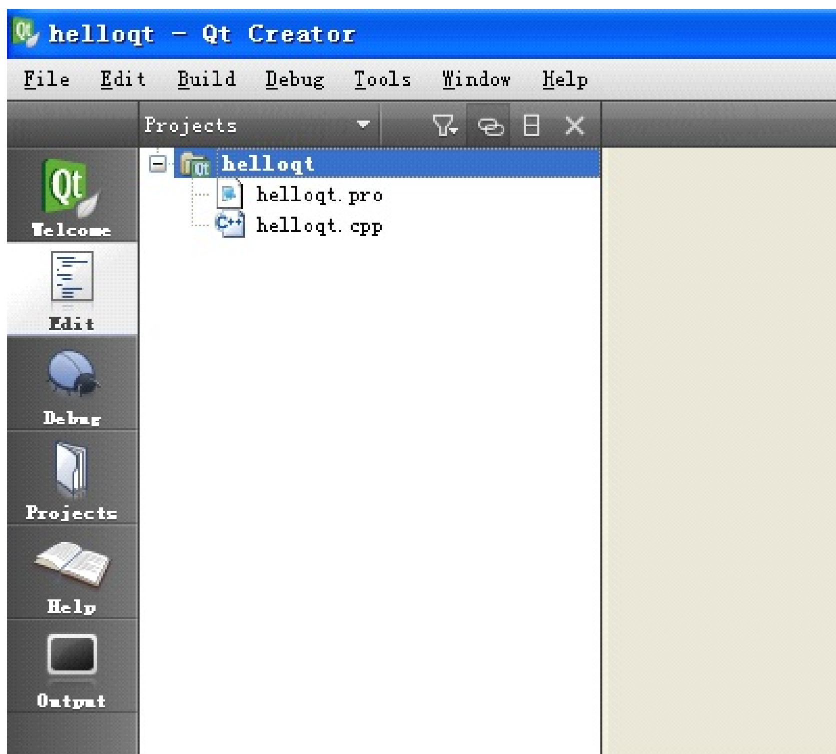


图 6-13 加入已经制作好的文件

小贴士：在 Qt Creator 中对添加已有文件采取的是“不拷贝添加”的方式，即添加完成 后，Qt4 的工程并没有把这个文件从原有的位置拷贝一份到工程目录里面，而是添加了它的 相对路径位置。这点可以通过添加文件完成后，查看 .pro 文件的内容看出来。在这一点上 Qt Creator 与 KDevelop 和 QDevelop 中有所不同，后两者都可以设置添加的方式是不拷贝 添加还是拷贝添加。

我建议还是自己先手动把文件放置在工程目录下面，然后再在 Qt Creator 中添加。一 方面便于统一管理工程文件，二来可以提高 Qt 应用程序的效率。

下面再介绍添加新文件的方法。

依次点击【File】→【New...】→【C++】→【C++ Source File】，如图 6-14 所示， 我们这里是要添加一个 C++实现文件，点击【OK】按钮，进入下一步的设置。

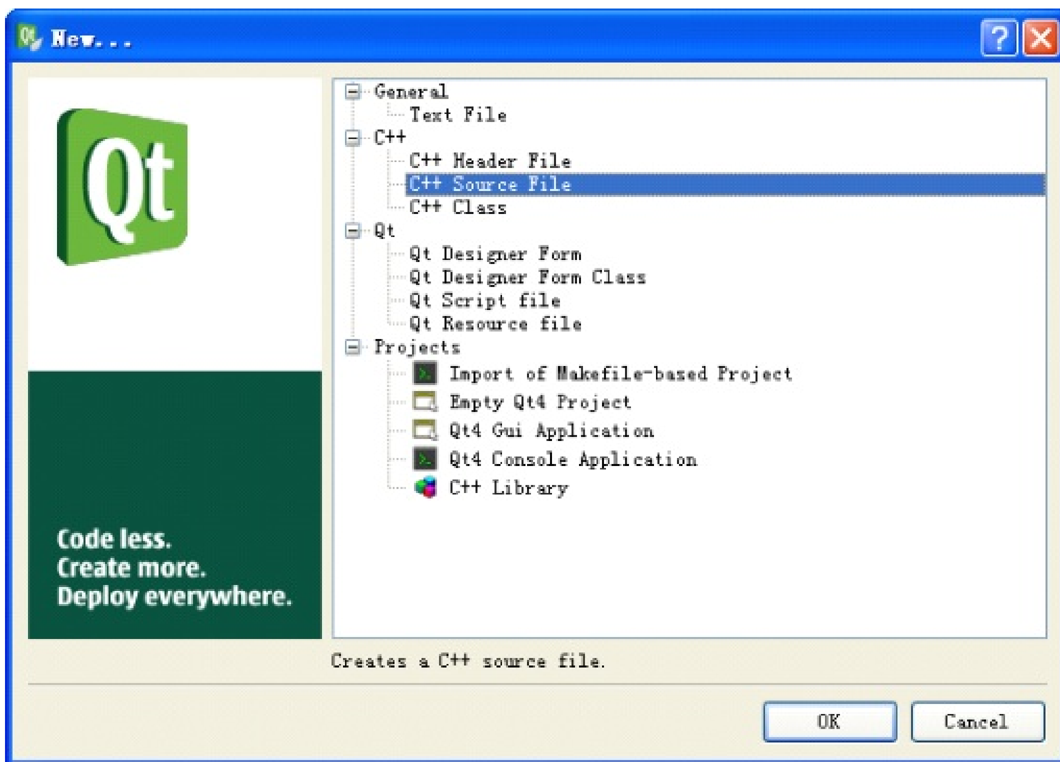


图 6-14 新建一个 C++实现文件

在接下来的如图 6-15 所示界面中，正确的输入文件名字，这里是 helloqt.cpp，并选 择工程的目录，然后点击 Next 按钮进入下一步。

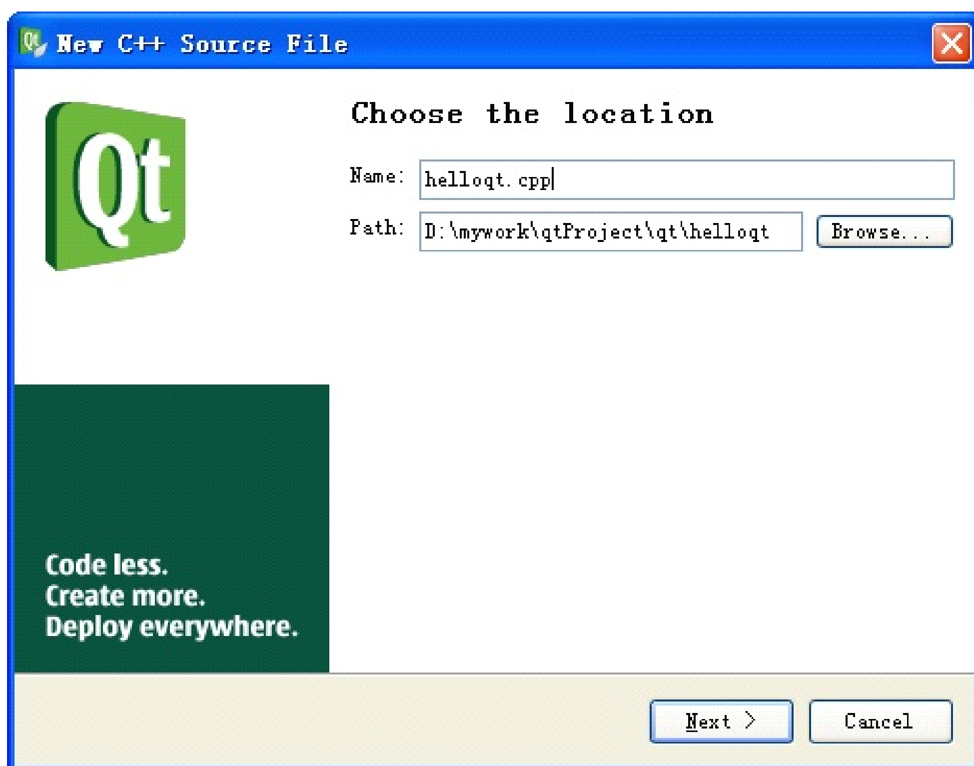


图 6-15 输入文件名

接下来，如图 6-16 所示，选择默认设置，把文件加入到 helloqt.pro 工程中，点击Finish 按钮。

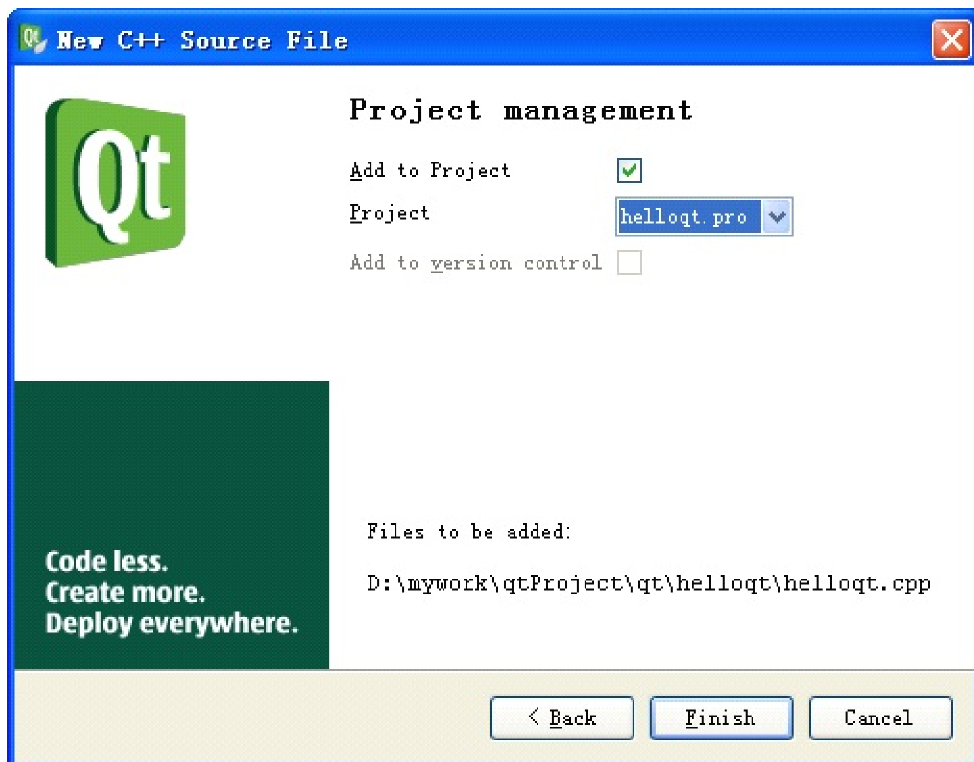


图 6-16 项目信息

这时，我们只是在工程里面加入了一个空的 C++实现文件，还需要添加内容。把 helloqt.cpp 源代码加入到如图 6-17 所示的屏幕右面的代码编辑区内。

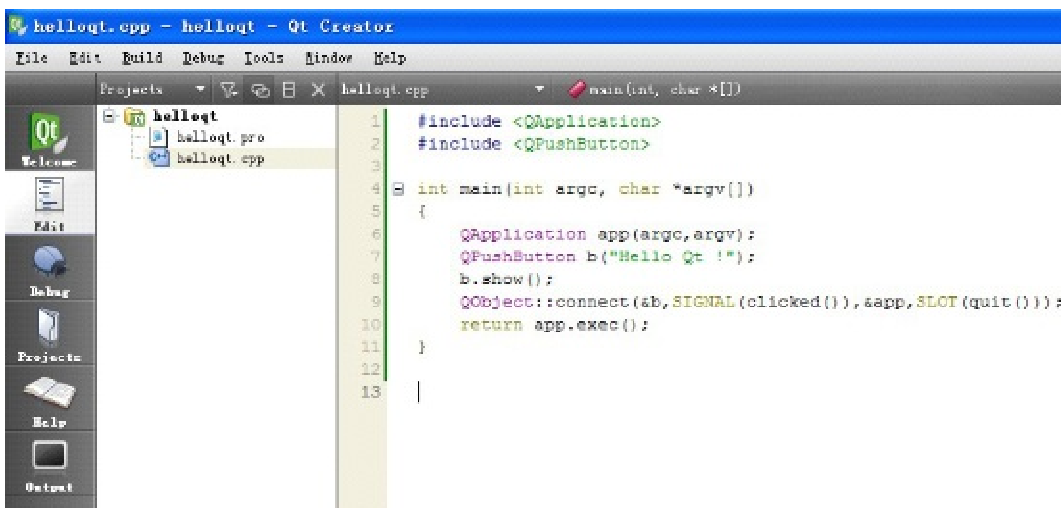


图 6-17 新文件加入后的样子

好了，编辑文件和设置工程的工作业已结束。接下来，就要开始编译源程序了，首先 运行 qmake。依次点击主菜单的【Build】→【Run qmake】,或者通过鼠标右键的上下文菜单来实现。

那么怎样才能知道 qmake 运行成功与否呢？Qt Creator 为我们提供了一个直观的查看 方式，当你运行 qmake 或者是 Build、Run 时，在 Qt Creator 的左下角会显示一个如图 6- 18 所示的运行状态进度条显示按钮，如果编译成功，它就会显示为满格的绿色；如果运行 不成功，或中途遇到 bug，它就会显示为不满格的红色，并且会提示 bug 和 warning 的数 量。



图 6-18 运行状态显示按钮

在这个进度条按钮上点击一下，就会显示出具体的状态信息，如图 6-19 所示，这个 Output 窗口显示了命令运行的过程，最后一句 ”Exited with code 0”表示成功。



图 6-19 运行 qmake 成功

接下来编译程序，依次点击主菜单的【Build】→【Build All Ctrl+Shift+B】,或者 通过鼠标右键的上下文菜单来实现，效果如图 6-20 所示。

```
Compile Output
Running build steps for project helloqt...
Configuration unchanged, skipping QMake step.
Starting: C:/Qt/2009.03/mingw/bin/mingw32-make.exe -w
mingw32-make: Entering directory `D:/mywork/qtProject/qt/helloqt'
C:/Qt/2009.03/mingw/bin/mingw32-make -f Makefile.Debug
mingw32-make[1]: Entering directory `D:/mywork/qtProject/qt/helloqt'
g++ -c -g -frtti -fexceptions -mthreads -Wall -DUNICODE -DQT_LARGEFILE_SUPPORT -DQT_DLL -DQT_GUI_LIB -DQT_CORE_LIB -
DQT_THREAD_SUPPORT -DQT_NEEDS_QMAIN -I"c:\Qt\2009.03\qt\include\QtCore" -I"c:\Qt\2009.03\qt\include\QtGui" -I"c:
\Qt\2009.03\qt\include" -I"c:\Qt\2009.03\qt\include\ActiveQt" -I"debug" -I"c:\Qt\2009.03\qt\mkspecs\win32-g++" -o
debug\helloqt.o helloqt.cpp
g++ -enable-stdcall-fixup -Wl,-enable-auto-import -Wl,-enable-runtime-pseudo-reloc -mthreads -Wl -Wl,-
subsystem,windows -o debug\helloqt.exe debug/helloqt.o -L"c:\Qt\2009.03\qt\lib" -lmingw32 -lqtmaind -lQtGuid4 -
lQtCored4
mingw32-make[1]: Leaving directory `D:/mywork/qtProject/qt/helloqt'
mingw32-make: Leaving directory `D:/mywork/qtProject/qt/helloqt'
Exited with code 0.
```

图 6-20 编译（build）成功

依次点击主菜单的【Build】→【Run Ctrl+R】,或者通过鼠标右键的上下文菜单来实 现，程序运行的效果与图 6-1 相同。

第 2 种方法的使用大致就是这样。不管是多么复杂的程序，使用它的流程都是与此类 似的。

6.2.5 第 3 种方法

这种方法也有两种常见的做法，一种是在单独启动的 Qt Designer 中设计程序界面， 制作完成后保存成.ui 文件（一种基于 XML 的文件格式），然后再在 IDE 中把它集成进来；

另外一种是直接在 IDE 中使用 Qt Designer，由于我们使用的 IDE 大都将 Qt Designer 的主要功能集成了进来，所以这两种做法其实区别不大，就看大家的喜好了。

下面仍以 Qt Creator 为例，使用这种方法时，一般遵循如下步骤：

- 创建窗体并在窗体中放置各种窗口部件
- 设置窗口部件的属性
- 对窗体进行布局设计
- 设置各个窗口部件的 Tab 顺序
- 创建信号和槽
- 连接信号和槽
- 编写代码（如果需要的话）
- 编译、链接、运行程序 1.建立新工程

依次点击【File】→【New...】→【Projects】→【Qt4 Gui Application】，如图 6- 14 所示，点击【OK】按钮进入下一步设置。

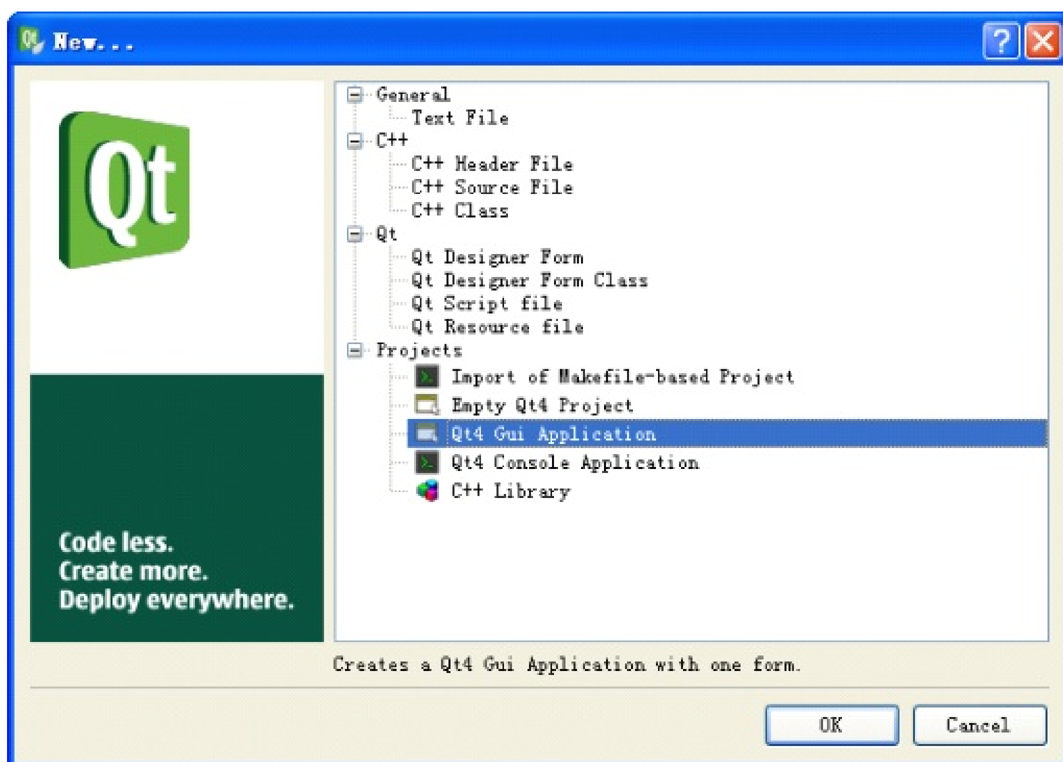


图 6-21 建立新工程

接下来仍旧是设置项目的名称和目录位置，正确设置即可，我们这里仍旧是 helloqt， 如图 6-22 所示，点击【Next】按钮进入下一步。

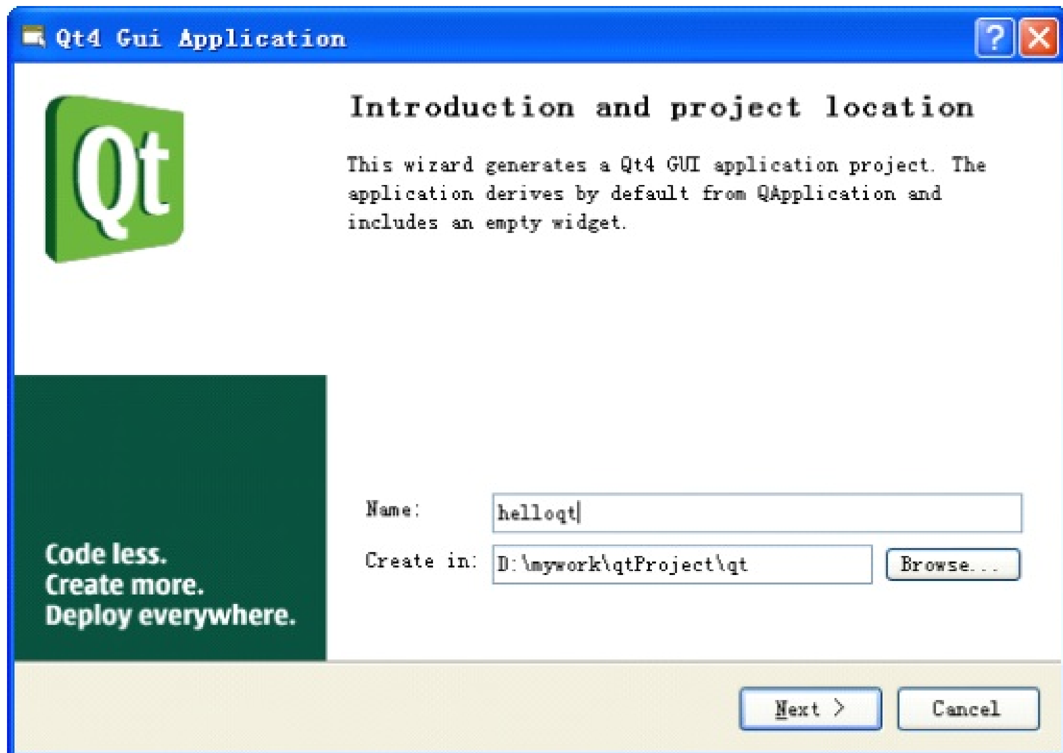


图 6-22 设置项目的名称和目录位置

接下来需要设置你的应用程序想要包含的 Qt 模块，当你选择了某个模块后，它的头文 件就会在程序中被自动包含了。如图 6-23 所示，其中 QtCore Module 和 QtGui Module 是 必需的，缺省已经选择了。我们这个程序并不需要包含其他的功能，选择缺省设置，点击【Next】按钮进入下一步。

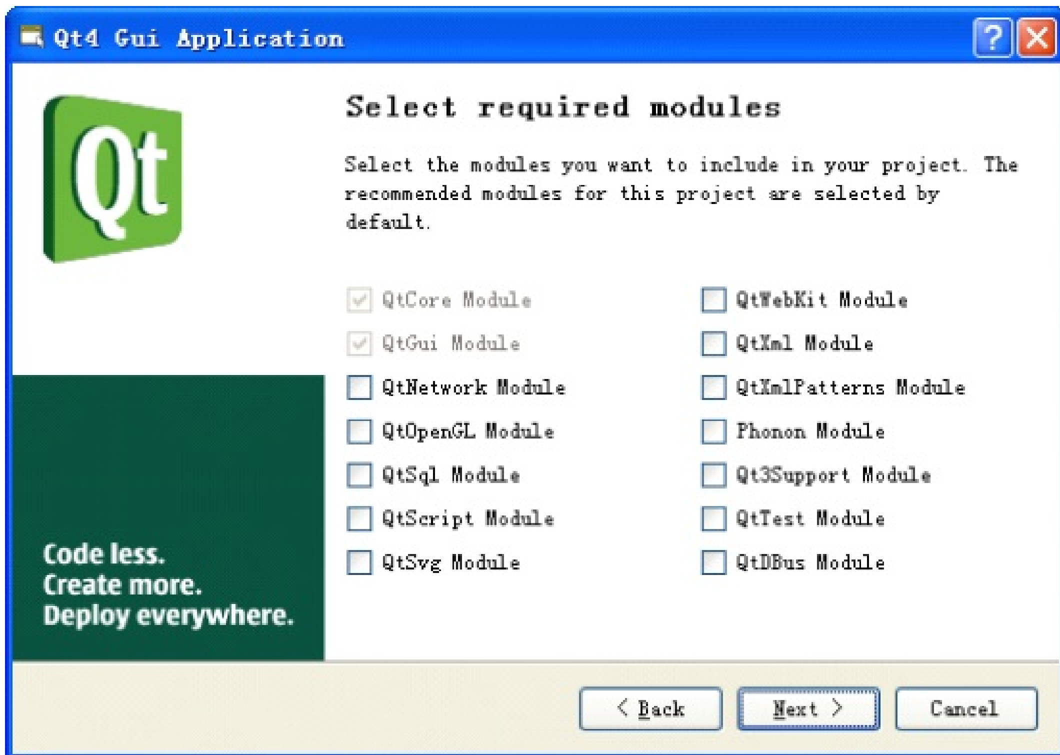


图 6-23 选择需要的 Qt 模块

下面我们需要设置应用程序的类型，这里是通过选择你建立的类的基类来体现的，共 有 3 种选择：QMainWindow、QWidget、QDialog。也就是说，我们的应用程序的骨干类是子 类化这 3 种类其中的一种。这里我们以选择基类为 QWidget，其它选项选择缺省值，如图6-24 所示，点击【Next】按钮进入下一步。注意这里需要将【Generate form】选项选上，否则将不会产生.ui 文件，这就和第 2 种方法是一样的了。

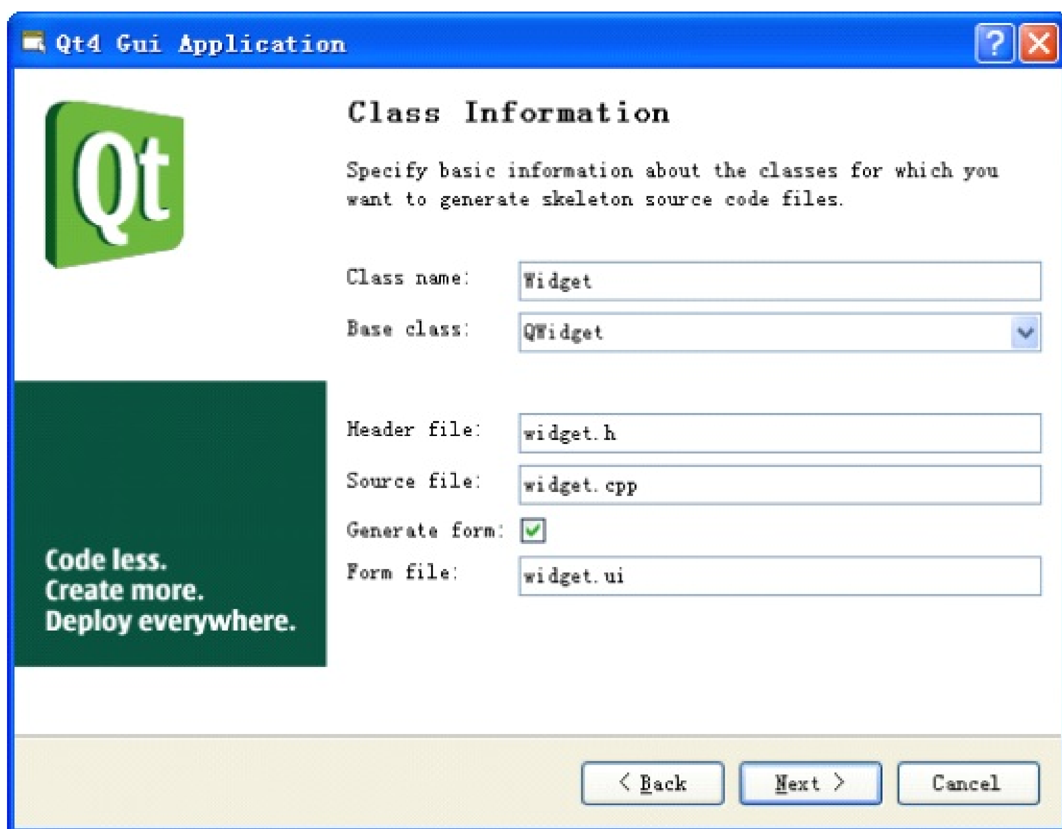


图 6-24 设置应用程序的类型

接下来展示的是要建立的工程的基本信息。我们的工程由主要由 5 个文件组成： main.cpp、widget.h、widget.cpp、widget.ui 和 helloqt.pro。其中 main.cpp 是主程序 文件，其中主要包含了 main()函数，前面我们已经讲到，它是应用程序的入口； widget.ui 是程序界面文件，它是基于 xml 格式的，它描述了界面的布局信息； widget.h 和 widget.cpp 是基于 widget.ui 产生的界面实体类；helloqt.pro 是工程文件，其中包含了 工程的基本信息，它能够被 qmake 所识别。点击【Finish】按钮，就完成了工程的设置。

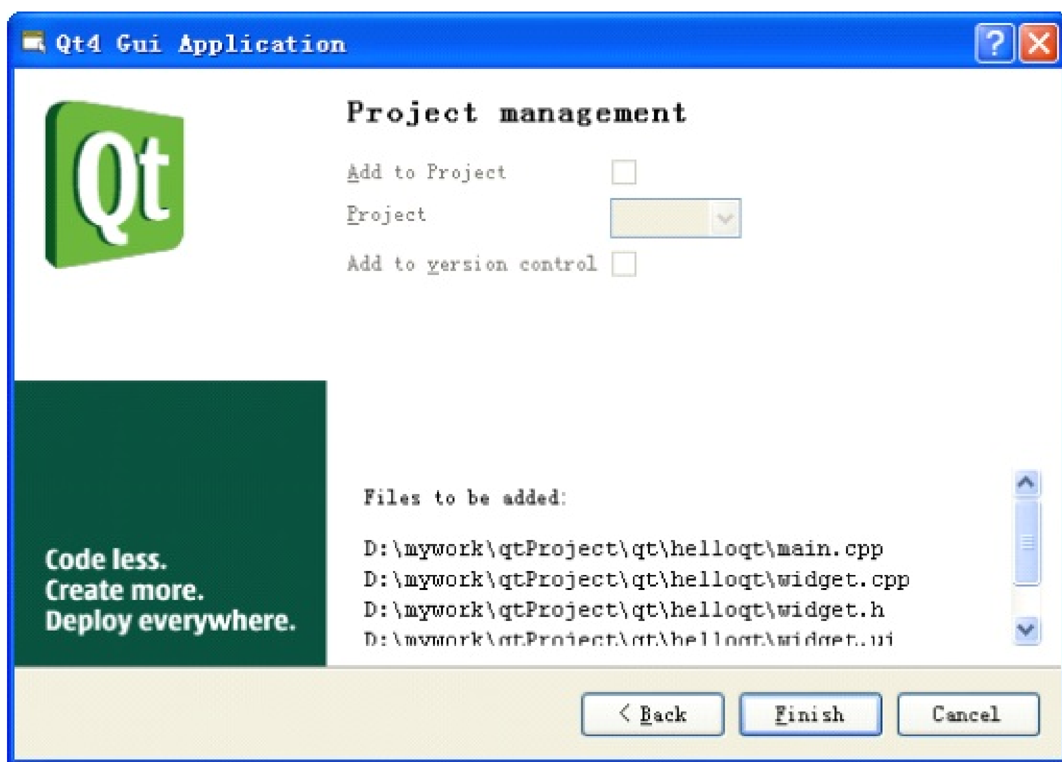


图 6-25 生成的工程信息

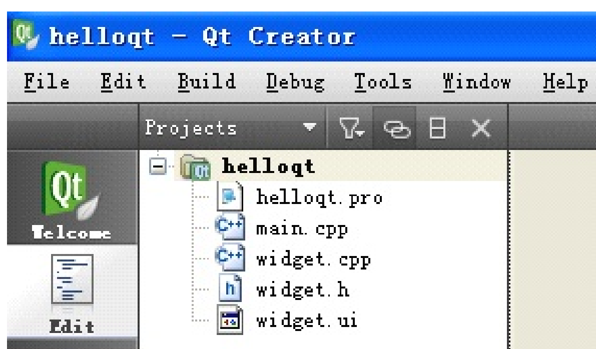


图 6-26 生成的工程概览

生成的工程如图 6-26 所示，如果想查看或修改某个文件的内容，在该文件上双击鼠标 左键即可在右面的代码浏览和编辑区内实现。

2.创建窗口部件并设置属性

用鼠标左键双击 widget.ui 文件，Qt Creator 将把 Qt Designer 打开并集成到框架 内，如图 6-27 所示，是不是很熟悉，与我们在第 5 章介绍的单独启动的 Qt Designer 并没 有什么不一样。

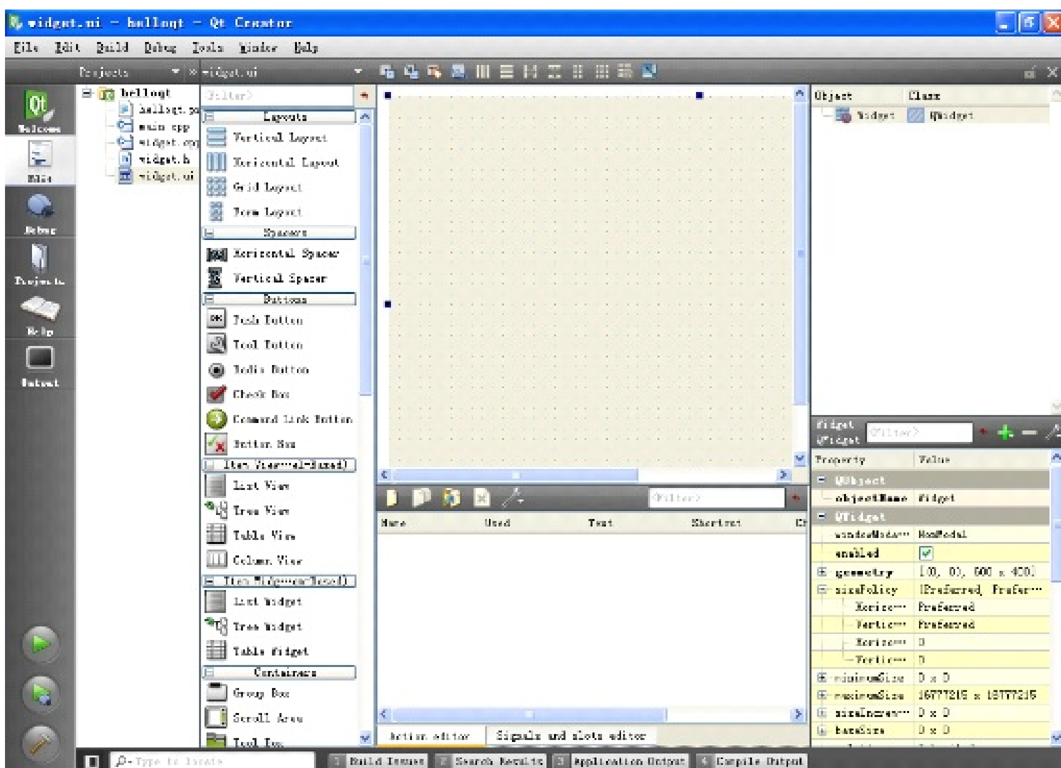


图 6-27 集成 Qt Designer 的 Qt Creator 设置 Widget 的 WindowTitle 属性为：Hello Qt!。在窗体上放置一个 Push Button 窗 体部件，并把它的 text 属性设置为：Hello Qt!。然后手动调整一下 Widget 和 Button 的 大小，不必太费力气，只要不是太难看就可。关于设置属性的方法在第 5 章中已经有详细的讲述，这里不再赘述。最后设置完成的界面如图 6-28 所示。



图 6-28 初步设置好的界面

3.设计布局和 Tab 顺序

由于程序的界面元素比较少，我们选择水平布局，再设置 Tab 顺序，完成后的界面如 图 6-29 所示。



图 6-29 设置布局和 Tab 顺序

4.创建并连接信号与槽

按照第 5 章介绍的方法，按下 F4 键，拖动鼠标左键，选择 pushButton 的 clicked()信 号，以及 Widget 的 close()槽，并连接它们，完成后的效果如图 6-30 所示。



图 6-30 创建并连接信号和槽

5.编译运行程序

依次执行 qmake、Build（或 Build All）、Run，程序运行后的效果如图 6-31 所示。



图 6-31 程序运行效果

估计读者朋友也看出来了，这个程序运行的效果与前面的不完全一样。这也正是我想 说明的，Qt Designer 并不是“为所欲为”的，比如这里我们就无法在其中设计出与第 1 种 手写代码完全一致的界面来，除非我们对其产生的代码再进行手动修改。

实际上，在进行工程开发时，尤其是面对复杂的大型应用程序时，程序员完全采用手 写代码的时候很少见，而完全使用 Qt Designer 设计界面的情形也不多，更多情况下往往 是将两者结合起来使用。但无论怎样，第 1 种方法都是最基本的，只有对 Qt 的基本原理很 清楚，对 qmake 的基本语法掌握的很熟练，才能够游刃有余的开发 Qt 应用程序。

6.3 几个重要的知识点

1.关于信号/槽

信号/槽提供了任意两个对象间通信的机制，是 Qt 区别于其它 GUI 库的最重要的特征 之一。传统的 GUI 库往往采用回调函数来实现对象间的通信，而 Qt 的信号/槽机制要更简 单灵活。每个 Qt 对象都包含特定的信号和槽以及相应的关联定义，当有事件发生或是对象 的状态发生改变时，对应的信号就被发射出来，与其相关联的槽被执行以响应该信号，完成 处理工作。

信号与槽通常采用如下连接方式：

```
connect(Object1,SIGNAL(signal),Object2,SLOT(slot));
```

式中，Object1 和 Object2 是两个对象，signal 是 Object1 发出的信号，slot 是用来响应 signal 信号的槽，它属于 Object2。SIGNAL()和 SLOT()是两个宏，它们是 Qt 语法中 的一部分。

关于信号/槽的深入内容，我们会在第 13 章为大家详细讲解。 2.构建 Qt 应用程序的的流程在构建 Qt 应用程序时，无论你是采用手写代码，还是使用 Qt Designer 设计界面，都会遵循这个一般的顺序，就是先声明所需使用的窗口部件，接下来设置它们的属性，然后再把这些窗口部件添加到布局中，布局会自动设置它们的位置和大小。接下来根据 Qt 的信号/槽原理，创建并连接使用到的信号和槽，通过窗口部件之间的通信就可以管理用户的交互 行为。最后就是程序的编译、链接和运行，基本的顺序是 qmake –project,qmake xxx.pro,make(具体的 make 命令与平台相关),然后运行你的程序。

3.编译 Qt 应用程序的方法

我们可以选择 3 种方式来编译 Qt 应用程序：

(1) 使用 qmake

这是最常用的方式，qmake 最为重要的作用是生成与平台无关的 .pro 文件，并以此来 生成与平台相关的 makefile。

(2) 使用集成开发环境

本质上使用 IDE 也是在使用 qmake。

(3) 使用第三方的编译工具

从理论上来说，任何第三方编译工具都可以用于 Qt 的应用程序开发中，但使用可以感 知 Qt（Qt-aware）的工具会比较容易一些。常见的第三方编译工具有 CMake、Boost.Build 和 Scons 等。它们的使用相对比较复杂，不推荐初学者使用。

4.qmake 工程文件的结构

这里我们看一下 Qt Creator 为我们生成的 helloqt.pro 这个工程文件的内容。

```
#-------------------------------------------------
#
# Project created by QtCreator 2009-08-16T22:31:08
#
#-------------------------------------------------
TARGET = helloqt
TEMPLATE = app
SOURCES += main.cpp\
widget.cpp
HEADERS += widget.h
FORMS += widget.ui
```

前 5 行，也就是用#号打头的内容是 qmake 工具自动添加的注释，这里它解释了改工程文件是使用 Qt Creator 创建的以及创建的时间。

变量 TARGET 描述了目标工程文件的名称，通常就是生成的应用程序的名字。

变量 TEMPLATE 描述了生成何种形式的 makefile 文件，有 5 种常见的模板：

app 建立一个 Qt 应用程序的 makefile lib 建立一个 Qt 应用库的 makefile

subdirs 建立一个子目录下目标文件的 makefile，子目录通过变量 SUBDIRS 指定（子 目录下若有工程文件也需要指出类型）

vcapp 为 Visual Studio 生成一个工程，可在 Windows 操作系统上使用 vclib 为 Visual Studio 生成库工程，可在 Windows 操作系统上使用

变量 SOURCES 选项告诉编译器，源代码文件的相对于工程文件 helloqt.pro 的位置以 及文件名字，本程序包含两个.cpp 文件，中间用\隔开。

同理，变量 HEADERS 告诉编译器头文件的路径，变量 FORMS 告诉编译器.ui 文件的路径。

有了上面这些信息，编译系统就知道该如何编译与平台相关的 makefile 文件了。

qmake 的语法很丰富，常用的一些我们会根据应用程序的情况陆续为大家讲解，附录 B 详细为大家介绍了 qmake 的用法。

6.4 问题与解答

问：初学 Linux 下的 Qt 编程，请教个大家一下关于 QT 程序的运行环境的问题。用 QT 编译好的程序，可不可以在不启动 X-Windows（KDE/GNOME）的情况下直接在 X 终端下运 行呢？

答：需要启动 X，至少要需要启动 X Server。因为 Qt 在 X11 上的运行机制是要依赖本 地原生的图形环境的。如果是 QT/E 的话只要启动了 Framebuffer 就可以了。

问：请问一下，我装好了 Dev C++ 和 QT4.2.2 自带的例子已经能够编译了，但是今天 编译一个例子就不能通过，不知道是什么问题，请指教。

我的程序是：

```
#include &lt;qapplication.h&gt;
#include &lt;qlabel.h&gt;
int main(int argc, char *argv[])
{
    QApplication app(argc, argv);
    QLabel *label = new QLabel("Hello Qt!", 0);
    app.setMainWidget(label);
    label-&gt;show();
    return app.exec();
}
```

编译后的出错信息是:

```
C:\b&gt;make
mingw32-make -f Makefile.Release
mingw32-make[1]: Entering directory `C:/b'
g++ -c -O2 -O2 -frtti -fexceptions -Wall -DUNICODE -DQT_LARGEFILE_SUPPORT -DQT_D
LL -DQT_NO_DEBUG -DQT_GUI_LIB -DQT_CORE_LIB -DQT_THREAD_SUPPORT -DQT_NEEDS_QMAIN
-I"C:/Qt/4.2.2/include/QtCore" -I"C:/Qt/4.2.2/include/QtCore" -I"C:/Qt/4.2.2/in
clude/QtGui" -I"C:/Qt/4.2.2/include/QtGui" -I"C:/Qt/4.2.2/include" -I"." -I"C:/Q
t/4.2.2/include/ActiveQt" -I"release" -I"." -I"..\Qt\4.2.2\mkspecs\default" -o r
elease\hello.o hello.cpp
hello.cpp: In function `int qMain(int, char**)':
hello.cpp:8: error: 'class QApplication' has no member named 'setMainWidget'mingw32-make[
*** [release\hello.o] Error 1
mingw32-make[1]: Leaving directory `C:/b'
mingw32-make: *** [release] Error 2
error: 'class QApplication' has no member named 'setMainWidget'
```

答：setMainWidget()这个方法是 Qt 3 中的方法，而 Qt 4 中是没有的。所以从你提供的出错信息上可以判断，你机器上安装的是 Qt 4 的某个版本，而你的程序的代码是用 Qt 3 写成的。由于从 Qt 3 到 Qt 4，其语法和类库等都发生了很大的变化，很多程序是不通用 的。所以你要么使用 Qt 4 编译 Qt 4 的程序，或者都换成是 Qt 3 的。

我的建议是，如果不是维护已有项目需要的话，最好还是学习 和使用 Qt4 吧，Qt 的网 站上说，Qt 3.3 系列将维护到 July 2007，也就是后续将不会提供支持了。

问：我初学 Qt 编程，请问如何入门，有什么建议吗？现在我还无处着手。 答：对于初学者而言，建议选定最新的 Qt 版本（4.5 以上），按照本书章节的顺序同时结合 Qt Assistant 和 Qt Demo 循序渐进的学习。注意刚开始时，可以照猫画虎，把书上的例子反复体会，然后最好是带着问题去学习，中间不要忘记多泡论坛和专家的博客（本书 附录中对 Qt 的论坛和博客有详细介绍）。

问：我最近开始学习 Qt,因为需要编写一个类似 Google 地图搜索的图形界面,但是不知 道怎么下手,各位有作过类似东东的吗,给小弟一点建议,应该怎么下手作啊,谢谢了,或者有 类似的源代码给我参考一下马?

答：这方面有一些思路可供参考。可以利用 Graphics View 显示地图。使用 QGraphics View 和 QGraphicsScene 进行坐标变换，使用 QGraphics View 的 scale()方法实现地图的 缩放。如果是初学的话，建议可以学习本书并结合 Qt Demo 的例子，这样提高比较快。

6.5 总结与提高

在这一章里面，我们通过 Hello Qt!这个例子，向大家介绍了 Qt 应用程序编译运行的 基本步骤，以及常见的 3 种方法。它们各有优缺点。如果你刚刚使用 Qt，我希望你通读本章，然后把 3 种方法都尝试一下，重点是 Qt 应用的开发流程，以及对信号/槽的初步理解。在这个过程中，你可能会遇到一些觉得很 “奇怪”的问题，别着急，它们往往来自 Qt 的安装和配置、Qt 头文件的包含以及与平台相关的基础知识等方面，大家可以回过头再看看前几章，往往会有很大的收获，这也就是 “温故而知新”的体现。

在后面的几章里，我们还会陆续为大家介绍对话框、主窗口、布局管理等方面的内 容，它们比本章的例子要复杂一些，但不管怎样，这其中不变的是 Qt 的基本原理以及 qmake 的语法和开发应用程序的流程，希望大家能够用心体会和掌握。

第 7 章 对话框

本章重点

- 了解对话框程序的作用
- 了解 QDialog 类及其子类的继承关系
- 掌握使用手写代码子类化 QDialog 类的方法
- 掌握结合 Qt Designer 创建对话框应用程序的方法
- 掌握常见的 Qt 标准内置对话框的使用方法
- 了解模态对话框和非模态对话框的区别并能够正确使用它们

绝大多数图形用户界面应用程序都带有一个由菜单栏、工具栏构成的主窗口以及一些 对主窗口进行补充的对话框。当然，也可以创建对话框应用程序，它可以通过执行合适的动 作来直接响应用户的选择（例如，一个计算器应用程序）。

这一章讲解如何使用 Qt 创建对话框。对话框为用户提供了许多选项和多种选择，允许 用户把选项设置为他们喜欢的变量值并从中做出选择。之所以把它们称为对话框，或者简称 为“对话”，是因为它们为用户和应用程序之间提供了一种可以相互 “交谈”的交互方式。

本章将首先完全用手写代码的方式创建第一个对话框程序，以便能够说明是如何完成 这项工作的。接下来将使用 Qt Designer 类实现相同的功能。使用 Qt Designer 比手写代 码要快得多，并且可以使不同的设计测试工作以及稍后对设计的修改工作变得异常轻松。然 后将为大家介绍 Qt 内建的对话框类的使用，最后讲解一下模态对话框和非模态对话框的区 别和用法。

7.1 QDialog 类

QDialog 类是对话框窗口的基类。对话框窗口是一个顶级窗口，通常用作短期任务，或 者是与用户的简短会话等场合。对话框可以分为模态对话框和非模态对话框。使用 QDialog 或其子类创建的对话框窗口通常都有一个返回值，有时候还包含了一些默认的按钮。一般情 况下，对话框窗口在其右下角都有一个用于控制其大小的伸缩手柄，在 Qt 应用程序中，这 一般可以通过调用 setSizeGripEnabled()方法来实现。

QDialog 是所有对话框类的基类，它继承自 QWidget，它的子类 有 QAbstractPrintDialog，QColorDialog，QErrorMessage， QFileDialog， QFontDialog，QInputDialog，QMessageBox， QPageSetupDialog，QPrintPreviewDialog，QProgressDialog，QWizard，以及来自 Qt3 的 Q3FileDialog，Q3ProgressDialog，Q3TabDialog，Q3Wizard。图 7-1 示意了 QDialog 及其 子类的继承关系。

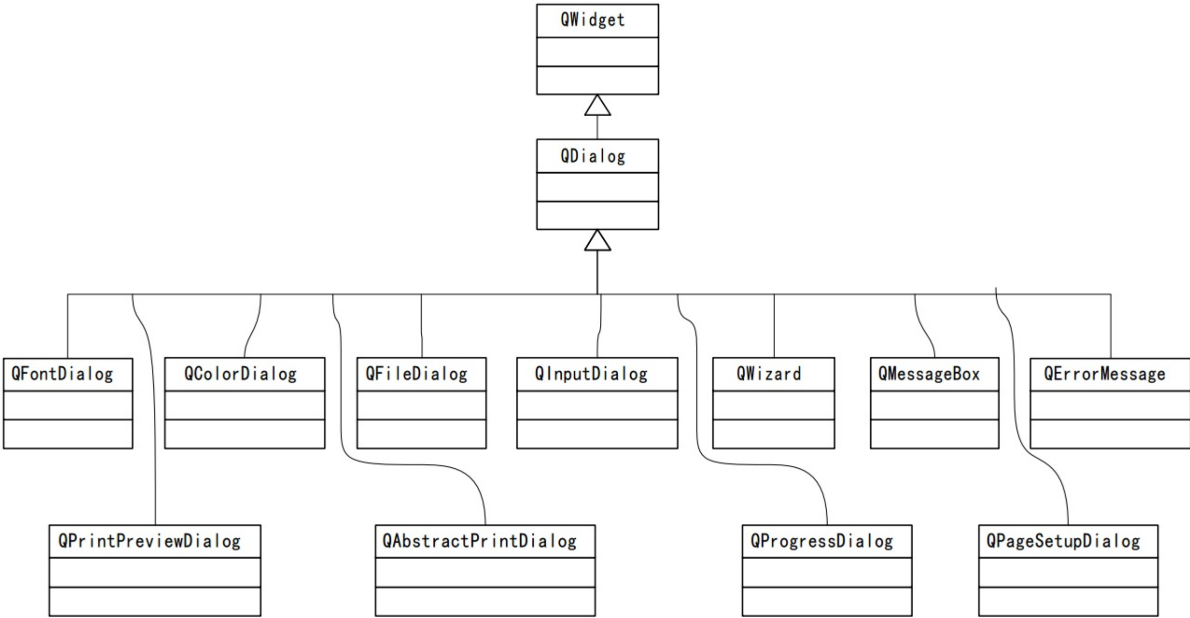


图 7-1 QDialog 类继承关系图

表 7-1 列举了 QDialog 子类的用途。

表 7-1 QDialog 子类说明

子类名	用途
QAbstractPrintDialog	提供打印机配置对话框的基本实现对话框
QColorDialog	提供指定窗体颜色的对话框
QErrorMessage	提供“错误提示”对话框
QFileDialog	提供选择文件或目录的对话框
QFontDialog	提供指定窗体的文字字体对话框
QInputDialog	提供标准输入对话框，可以方便的输入各种值
QMessageBox	提供一个模态对话框用于提示用户信息或要求用户回答问题
QPageSetupDialog	提供一个用于打印机页面设置的对话框
QPrintPreviewDialog	提供一个预览和调整打印机页面布局的对话框
QProgressDialog	提供一个长进程操作的进度回馈对话框
QWizard	提供一个“向导程序”的框架

在实际应用中，我们经常会用到 QColorDialog、QFileDialog、QInputDialog、QMessageBox 等这些内置的标准对话框，在第 7.5 节里面我们将通过实例具体讲解它们的使用要领。

7.2 子类化 QDialog

创建基于对话框的应用程序主要是使用子类化 QDialog 的方法。在本节，我们采用这个方法创建一个稍微复杂的实例-可扩展对话框。

可扩展对话框通常只显示简单的外观，但是它还有一个切换按钮（toggle button），可以让用户在对话框的简单外观和扩展外观之间来回切换。可扩展对话框通常用于试图同时满足普通用户和高级用户需要的应用程序中，这种应用程序通常会隐藏那些高级选项，除非用户明确要求看到它们。

这个对话框是一个用于电子制表软件应用程序的排序对话框（Sort 对话框），在这个对话框中，用户可以选择一列或多列进行排序。在这个简单外观中，允许用户输入一个单一的排序键，而在扩展外观下，还额外提供了两个排序键。More 按钮允许用户在简单外观和扩展外观之间切换。该实例的运行效果如图 7-2 所示。

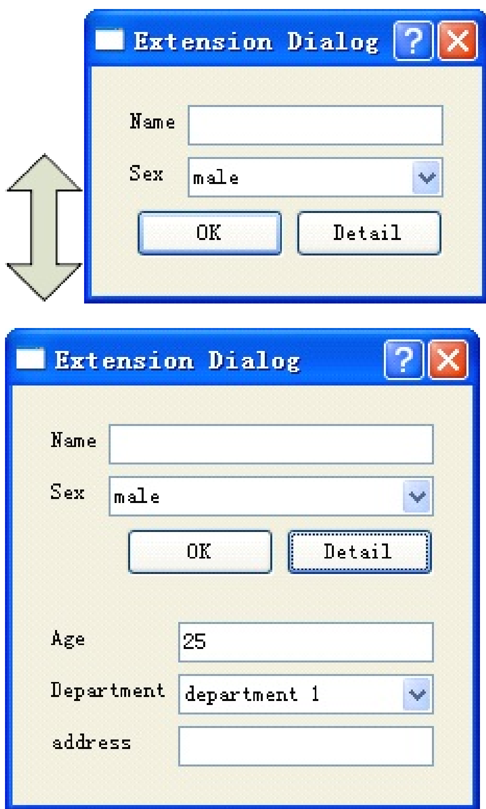


图 7-2 实例运行效果

该实例名为 extensionDlg。共有以下原生源文件：工程文件 extensionDlg.pro，主程序文件 main.cpp，对话框类 ExtensionDlg 的头文件 extensionDlg.h，实现文件 extensionDlg.cpp。

在第 6 章中我们向大家介绍了创建 Qt 应用程序的基本方法，这里我们采用 Qt Creator作为 IDE，并使用完全手写的方式完成程序的界面布局和构建。

首先当然是在 Qt Creator 中创建这个名为 extensionDlg 的项目，类型是 Empty Qt4 Project。然后在其中依次加入对话框类 ExtensionDlg 的头文件 extensionDlg.h，实现文件 extensionDlg.cpp，主程序文件 main.cpp 以及工程文件 extensionDlg.pro。这些文件的内容， 可以在 Qt Creator 的代码编辑器中完成编辑。

我们先看一下对话框类 ExtensionDlg 的头文件的内容。

```
#ifndef EXTENSIONDLG_H
#define EXTENSIONDLG_H
#include &lt;QtGui&gt;
class ExtensionDlg : public QDialog
{
    Q_OBJECT
public:
    ExtensionDlg();
    void initBasicInfo();
    void initDetailInfo();
    public slots:
    void slot2Extension();
private:
    QWidget *baseWidget;
    QWidget *detailWidget;
};
```

第 1 行引入 QtGui 模块的头文件

第 2 行声明我们的自定义对话框类 ExtensionDlg 单公有继承自 QDialog。

第 3 行加入 Q_OBJECT 宏，程序中用到诸如信号/槽等 Qt 核心机制的时候，都要加入这 个宏。

第 4-7 行声明了构造函数和初始化基础信息和扩展信息的函数。

第 8-9 行声明公有槽 slot2Extension()，它在用户点击【Detail】按钮时被触发。

第 10-12 行声明两个私有成员变量 baseWidget 和 detailWidget，它们都是 QWidget 的 实例，分别代表伸缩前后的对话框窗体。

再来看看 ExtensionDlg 的实现文件。

```
#include "extensionDlg.h"
ExtensionDlg::ExtensionDlg()
{
    setWindowTitle(tr("Extension Dialog"));
    initBasicInfo();
    initDetailInfo();
    QVBoxLayout *layout = new QVBoxLayout;
    layout-&gt;addWidget(baseWidget);
    layout-&gt;addWidget(detailWidget);
    layout-&gt;setSizeConstraint(QLayout::SetFixedSize);
    layout-&gt;setSpacing(6);
    setLayout(layout);
}
void ExtensionDlg::initBasicInfo()
{
    baseWidget = new QWidget;
    QLabel *nameLabel = new QLabel(tr("Name"));
    QLineEdit *nameEdit = new QLineEdit;
    QLabel *sexLabel = new QLabel(tr("Sex"));
    QComboBox *sexComboBox = new QComboBox;
    sexComboBox-&gt;addItem(tr("male"));
    sexComboBox-&gt;addItem(tr("female"));
    QPushButton *okButton = new QPushButton(tr("OK"));
    QPushButton *detailButton = new QPushButton(tr("Detail"));
    connect(detailButton,SIGNAL(clicked()),this,SLOT(slot2Extension()));
    QDialogButtonBox *btnBox = new QDialogButtonBox(Qt::Horizontal);
    btnBox-&gt;addButton(okButton,QDialogButtonBox::ActionRole);
    btnBox-&gt;addButton(detailButton,QDialogButtonBox::ActionRole);
    QFormLayout *formLayout = new QFormLayout;
    formLayout-&gt;addRow(nameLabel,nameEdit);
    formLayout-&gt;addRow(sexLabel,sexComboBox);
    QVBoxLayout *vboxLayout = new QVBoxLayout;
    vboxLayout-&gt;addLayout(formLayout);
    vboxLayout-&gt;addWidget(btnBox);
    baseWidget-&gt;setLayout(vboxLayout);
}
void ExtensionDlg::initDetailInfo()
{
    detailWidget = new QWidget;
    QLabel *ageLabel = new QLabel(tr("Age"));
    QLineEdit *ageEdit = new QLineEdit;
    ageEdit-&gt;setText(tr("25"));
    QLabel *deptLabel = new QLabel(tr("Department"));
    QComboBox *deptComboBox = new QComboBox;
    deptComboBox-&gt;addItem(tr("department 1"));
    deptComboBox-&gt;addItem(tr("department 2"));
    deptComboBox-&gt;addItem(tr("department 3"));
    deptComboBox-&gt;addItem(tr("department 4"));
    QLabel *addressLabel = new QLabel(tr("address"));
    QLineEdit *addressEdit = new QLineEdit;
    QFormLayout *formLayout = new QFormLayout;
    formLayout-&gt;addRow(ageLabel,ageEdit);
    formLayout-&gt;addRow(deptLabel,deptComboBox);
    formLayout-&gt;addRow(addressLabel,addressEdit);
    detailWidget-&gt;setLayout(formLayout);
    detailWidget-&gt;hide();
}
void ExtensionDlg::slot2Extension()
{
    if (detailWidget-&gt;isHidden())
    {
        detailWidget-&gt;show();
    }
    else
    {
        detailWidget-&gt;hide();
    }
}
```

第 1-9 句是构造函数中的内容。

第 1 句设置应用程序的标题。

第 2 句调用 initBasicInfo()函数，初始化基本信息船窗体。 第 3 句调用 initDetailInfo()函数，初始化扩展信息窗体。 第 4-9 句设置窗体的布局。

第 4 句定义一个垂直布局类实体 layout。

第 5、6 两句，分别将 baseWidget 和 detailWidget 加入到布局中。

第 7、8 两句对于布局管理来说是非常常见的用法，必须熟练掌握。其中 setSizeConstraint()函数用于设置窗体的缩放模式，其默认取值是 QLayout::SetDefaultConstraint。这里取参数值为 Qlayout::SetFixedSize 是为了使窗体 的大小固定，不可经过鼠标拖动而改变大小；如果不这样设置，当用户再次点击 【Detail】按钮时，对话框将不能恢复到初始状态。 setSpacing()函数用于设置位于布局之中的窗口部件之间的间隔大小。

这里暂时不对布局管理的相关内容展开讲解，在《布局管理》这一章中将有详细的介绍。

第 9 句将刚刚设置好的布局应用加载到窗体上。

第 10-29 句是 initBasicInfo()函数体的内容。

第 10 句实例化 baseWiaget，注意 baseWidget 是全局变量。

第 11-18 句，依次定义窗体中的部件，注意在输入字符时，前面都加上了 tr()函数。 第 19 句对于整个程序来说是关键，它使用信号 /槽机制连接了 detailButton 的单击信号和窗口类 ExtensionDlg 的 slot2Extension()函数，这就使得整个对话框变得可伸缩。

第 20-22 行示范了 QDialogButtonBox 类的用法，它用于创建一个符合当前窗口部件样 式的一组按钮，并且它们被排列在某种布局之中。在 Qt Designer 中，最为常见的用法是 从窗口部件盒里面把默认的那个 QDialogButtonBox 窗口部件拖到界面上来，不过显然这并 不如使用代码来得方便。

第 20 行的 Qt::Horizontal 实参表示创建水平方向的按钮组合。

第 21、22 两行把两个按钮加入到这个组合之中。其中的 QDialogButtonBox::ActionRole 参数表示创建的按钮具有实际的功能，单击它可以引起对话框的某种变化。该参数可以有很 多不同的值，使得这些按钮具有不同的功能，它们都被包含在 QDialogButtonBox::ButtonRole 这个枚举值之中。

第 23-29 行设置窗体的布局。窗体的顶级布局是一个垂直布局，而其中嵌套了一个表 单布局。

第 23-25 行是设置表单布局的常用方法，表单布局常用于窗体界面元素可以整齐的分 成两列的情况。addRow()方法用于向布局中加入整行的界面元素。

第 26-29 行定义窗体的顶级布局，并将其两个元素 formLayout 和 btnBox 依次加入其中。

小贴士：布局也是一种窗口部件，认识到这一点很关键。

第 30-47 行定义了 initDetailInfo()函数。其实现过程与 initBasicInfo()函数大同 小异。只是需要注意第 47 行，正是由于对 detailWidget 调用了 hide()函数，才使得程序 初始时，显示的是基本信息，而将扩展信息隐藏了起来。 hide()函数用于将窗口隐藏起来， 它是 Qt 默认的槽函数之一，其原型如下：

```
void QWidget::hide () [slot]
```

它的作用等同于调用 setVisible(false)函数。

第 48-55 行定义了 slot2Extension()函数。前面讲到，它是我们自定义的槽函数，在 点击【Detail】按钮时，将被触发。它的内容很简单，就是判断扩展窗口是否被隐藏，如果被隐藏，就显示它；否则就隐藏它。 isHidden()函数用于判断窗体的显示窗体的显隐状态。

然后就可以像下面这样书写主函数。第 1 句是必需的。第 2-4 句定义 ExtensionDlg 的 对象，并模态的显示这个窗体。

```
#include &lt;QApplication&gt;
#include "extensionDlg.h"
int main(int argc, char * argv[])
{
    QApplication app(argc,argv);
    ExtensionDlg exDlg;
    exDlg.show();
    return app.exec();
}
```

最后就是书写项目文件 ExtensionDlg.pro 了。

```
TEMPLATE = app
TARGET =
DEPENDPATH += .
INCLUDEPATH += .
# Input
HEADERS += extensionDlg.h
SOURCES += extensionDlg.cpp main.cpp
```

第 1 句，表示程序的类型是 app。

第 2 句中 TARGET 指定可执行文件或库的基本文件名，其中不包含任何的扩展、前缀或 版本号，默认的就是当前的目录名。

第 4 句中，INCLUDEPATH 指定 C++编译器搜索全局头文件的路径。 第 5 句是注释。

第 6 句中，HEADERS 指定工程的 C++头文件（.h）。多个头文件的情况下，用空格隔 开。

第 7 句中,SOURCES 指定工程的 C++实现文件（.cpp），多个头文件的情况下，用空格隔开。

qmake 的语法规则是比较简明的，详细情况请参阅附录。 好了，到此我们就完成了使用子类化的方法创建自定义对话框窗口程序。这其中有几点还想再强调一下， 一是要明白所谓子类化自 QDialog 类就是指通过继承自 QDialog 类来创建自定义的对话框类；二是要知道如果程序中用到了 Qt 的核心机制（如信号/槽）， 不 要忘记在类的声明处的第一行加入 Q_OBJECT 宏；三是要掌握判断和控制显隐对话框的方法；四是掌握定义窗口界面元素的方法（先定义， 再设置其属性）；五是了解布局管理的基本方法（后面还会学到）；六是要掌握书写工程文件的基本方法。七是掌握 Qt Creator 的基本用法。

在下一节，我们将采用 Qt Designer 完成这个程序界面的布局，大家将会了解到，有 时候使用 Qt Designer 设计界面会方便一些。

7.3 快速设计对话框

（续上面的例子，使用 Qt Designer 设计界面） 在这一节中将通过使用 Qt Designer 来创建与上一节相同的可扩展对话框， 并且使用 Qt Creator 作为 IDE 来管理这个工程。

我们将在 Qt Designer 中创建这个对话框的扩展外观，并且在运行时根据需要隐藏 扩 展信息。这个窗口看起来有些复杂，但在 Qt Designer 中可以轻易的完成它。

第 1 步，新建 Qt Creator 工程。

首先启动 Qt Creator，依次单击菜单项【File】->【New】，在弹出的对话框中选择工 程类型为【Empty Qt4 Project】，如图 7-3 所示。点击【Next】按钮进入下一步。

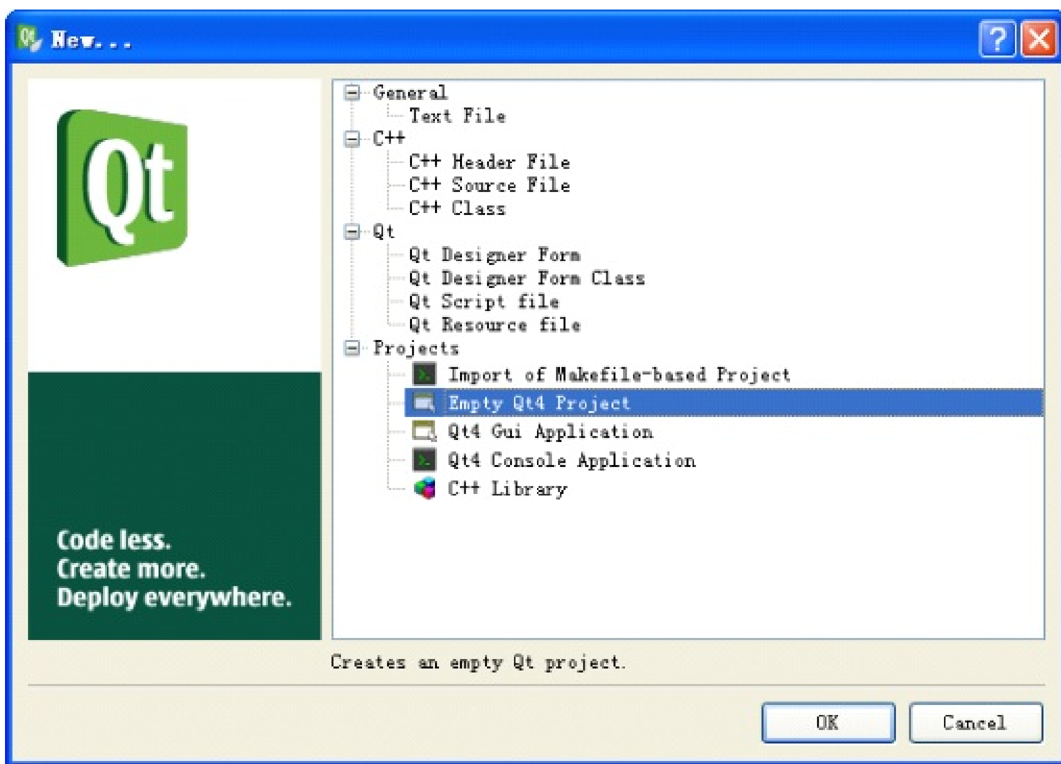


图 7-3 创建新工程

然后，像图 7-4 那样设置工程名称和存放的路径（根据自己的情况调整）。点击【Next】按钮进入下一步。

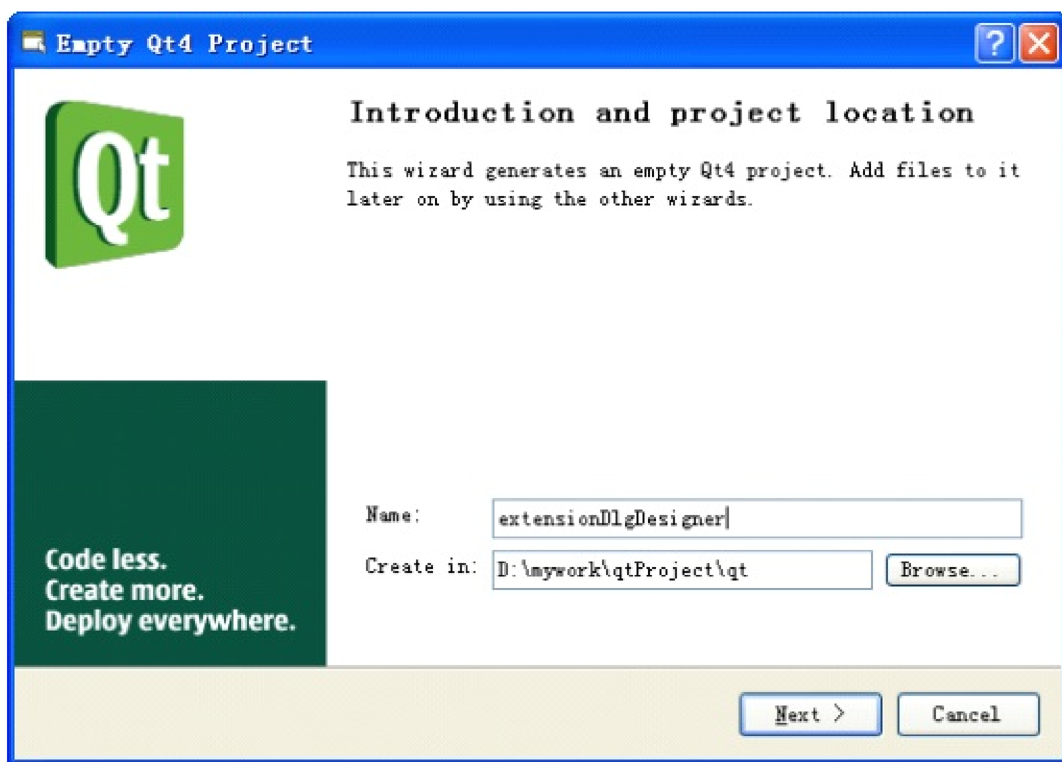


图 7-4 设置项目名称和存放的位置

在弹出的如图 7-5 所示的对话框中，点击【Finish】按钮完成工程的创建。

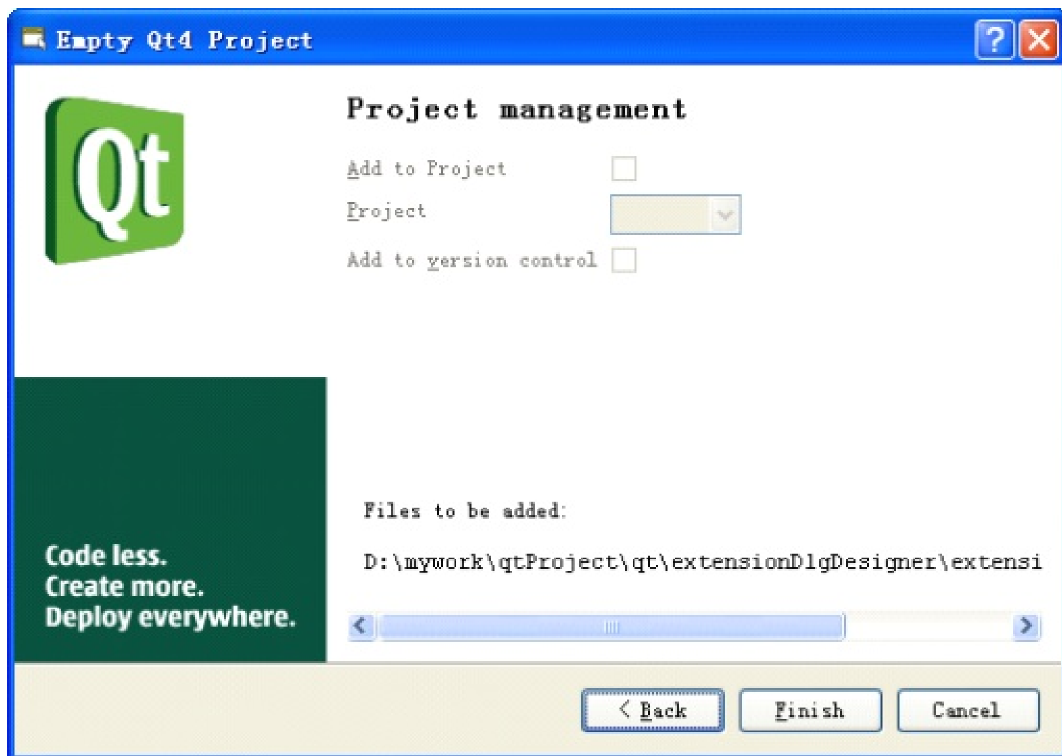


图 7-5 完成工程的创建

第 2 步，启动 Qt Designer，新建窗体。

依次单击菜单【File】->【New Form】，如图 7-6 所示，在【新建窗体】对话框中选择【Dialog without Buttons】模板。

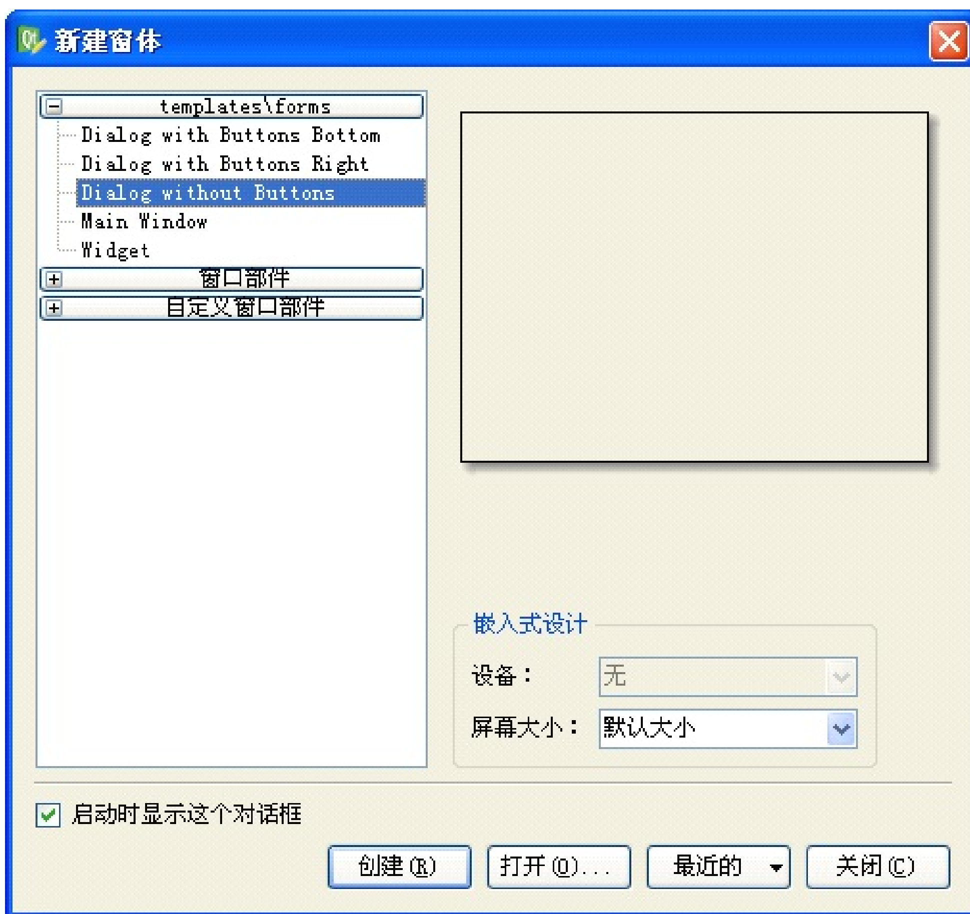


图 7-6 第 2 步 新建窗体

第 3 步，创建【基本信息】窗体的界面元素。

创建一个 GroupBox 部件，使它足够大，将它的 objectName 属性修改为 basicGroupBox，将它的 title 属性设置为 Basic。

接下来在 basicGroup 部件的内部创建下述的各个部件。 创建【Name】和【Sex】标签，使它们上下排列，将它们的 objectName 属性分别设置为 nameLabel 和 sexLabel，将它们的 text 属性分别设置为 Name 和 Sex。

再创建 Name 标签对应的编辑框，从窗口部件盒中拖出 LineEdit 控件。把它的 objectName 属性设置为 nameLineEdit。

创建 Sex 标签对应的组合框，从窗口部件盒中拖出 ComboBox 窗口部件，把它的 objectName 属性设置为 sexComboBox。

接下来为 Sex 标签对应的组合框创建列表项，方法是双击该组合框，如图 7-7 所示，在【编辑组合框】对话框中添加。



图 7-7 编辑组合框

创建【OK】按钮并把它拖放到【Sex】标签的下方。将它的 objectName 属性修改为 okButton，并将它的 defaults 属性设置为 true。

创建【Detail】按钮并将它拖放到 OK 按钮的水平右方，将它的 objectName 属性修改 为 detailButton,并将它的 defaults 属性设置为 true。

接下来，再创建一个垂直分隔符并把它拖放到 【Basic】分组框的下方。 不必刻意调整各个窗口部件的位置，只需大致排列一下即可。布置好后的情形如图 7-8所示。

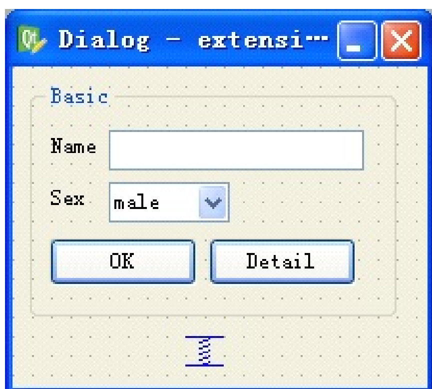


图 7-8 第 3 步 创建基本信息窗体的界面元素

第 4 步，设置【基本信息】窗体的布局。

单击【OK】按钮，按下 Shift 键后再单击【Detail】按钮，然后单击工具栏上的【Lay Out Horizontally】按钮，设置一个水平布局。

选中【Name】和【Sex】标签，以及它们对应的 LineEdit 窗口部件，然后单击工具栏 上的【Lay Out In a Form】按钮，设置为表单布局。

最后点击【Basic】分组框，然后单击工具栏上的【Lay Out Vertically】按钮，设置一个垂直布局。

设置到这里的界面情形如图 7-9 所示。

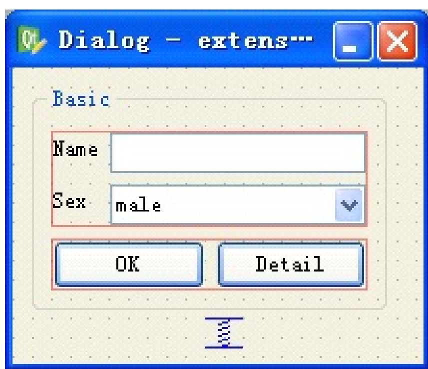


图 7-9 第 4 步 基本信息界面布局

第 5 步，创建【扩展信息】窗体界面元素。 拖动窗体的右下角，使其足够高和宽，以便能够容纳所有的界面元素。 创建一个 GroupBox 部件，使它足够大，将它的 objectName 属性修改为 extensionGroupBox，将它的 title 属性设置为 Extension。

像创建【基本信息】窗体界面元素那样，依次创建 Age 标签、Department 标签、 Address 标签以及和它们对应的 LineEdit 和 ComboBox，并把它们放入到 extensionGroupBox 分组框之中。

为 Department 标签对应的 ComboBox 添加列表项。 这样设置后的界面情形如图 7-10 所示。

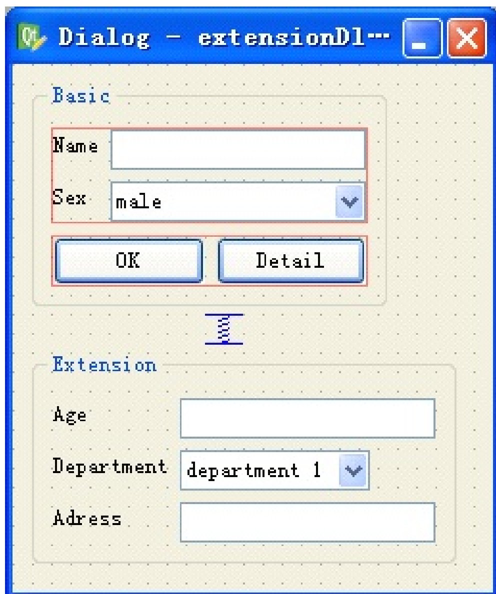


图 7-10 第 4 步 设置【扩展信息】窗体界面元素。

第 6 步，设置【扩展信息】窗体布局。

选中【Age】、【Department】和【Adress】标签，以及它们对应的编辑框、组合框窗 口部件，然后单击工具栏上的【Lay Out In a Form】按钮，设置为表单布局。

选中【Extension】分组框，然后单击工具栏上的【Lay Out Vertically】按钮，设置 为垂直布局，如图 7-11 所示。

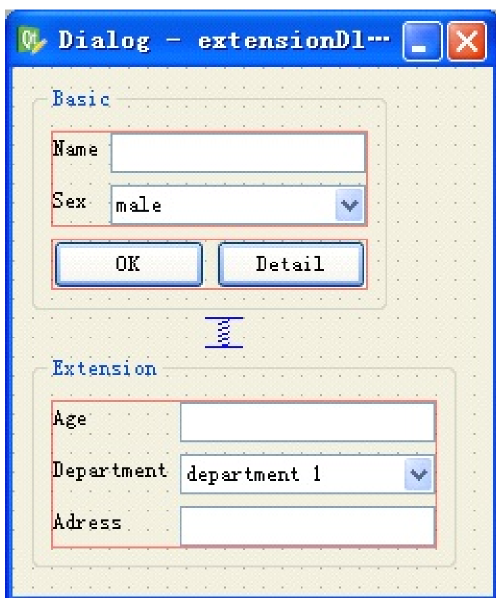


图 7-11 第 6 步 设置【扩展信息】信息窗体布局

第 7 步，设置窗体的顶级（Top Level）布局。 用鼠标左键点击窗体，注意不要选中任何窗口部件，然后单击工具栏上的 【Lay Out Vertically】按钮，设置窗体的顶级布局为一个垂直布局，如图 7-12 所示。

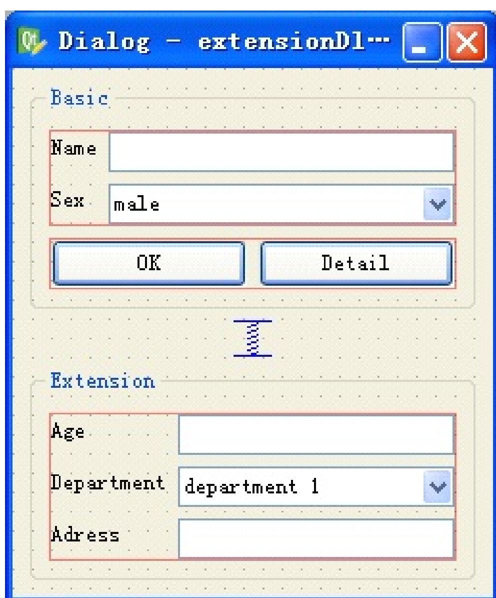


图 7-12 第 7 步 设置窗体的顶级布局

小贴士：如果没能生成你所希望的那种布局效果，或者是不小心做错了，那么总是可以随时先通过单击菜单项【Edit】->【Unto】，或者是【Form】->【Break Layout】，然后再重 新放置这些要摆放的窗口部件，最后试着对它们重新布局，直到满意为止。

第 8 步，设置 Tab 顺序。

单击菜单项【编辑】->【编辑 Tab 顺序】，或者点击工具栏上对应的按钮，像图 7-13 所示那样编辑 Tab 顺序。

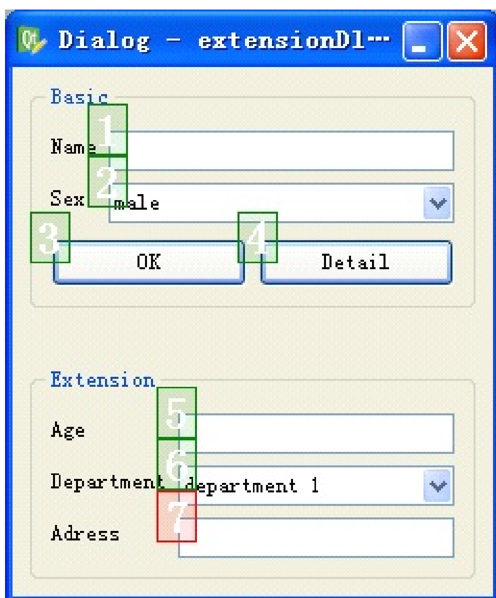


图 7-13 第 8 步 设置 Tab 顺序

第 9 步，设置信号/槽

按下 F3 键，离开 Tab 顺序设置模式，进入窗口编辑模式。 现在，窗体设计已经完成，可以开始着手设置一些信号 /槽的连接来实现窗体的功能了。Qt Designer 允许我们在构成同一窗体的不同部分内的窗口部件之间建立连接。我们需要建立两个连接。

单击菜单项【编辑】->【编辑信号/槽】，或者按下 F4 键，进入 Qt Designer 的信号- 槽连接设置模式。窗体中各个窗口部件之间的连接用蓝色箭头表示，如图 7-14 所示。并且 它们也会同时在 Qt Designer 的信号/槽编辑器窗口中显示出来。

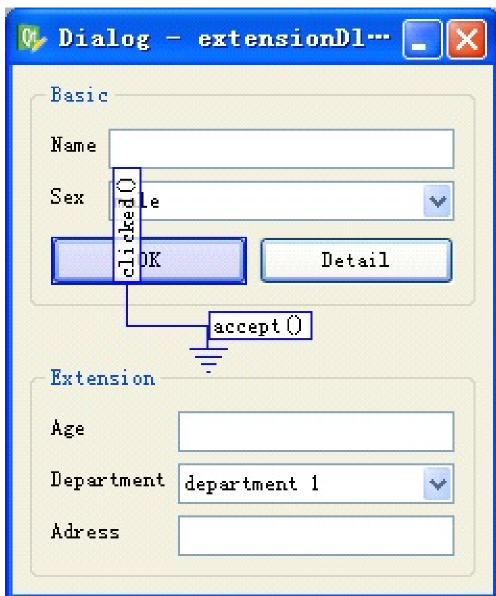


图 7-14 窗体中各个部件的连接是蓝色的线

小贴士：要在两个窗口部件之间建立连接，可以单击作为发射器的窗口部件并且拖动所产 生的红色箭头线到作为接收器的窗口部件上，然后松开鼠标按键。这时会弹出一个对话框， 可以从中选择建立连接的信号和槽。

要建立的第一个连接位于 okButton 按钮和窗体的 accept()槽之间。把从 okButton 按 钮开始的红色箭头线拖动到窗体的空白区域，然后松开按键，这样会弹出图所示的设置连接 对话框（Configure Connection dialog）。从该对话框中选择 clicked()作为信号，选择 accept()作为槽，然后单击 OK 按钮，如图 7-15 所示。

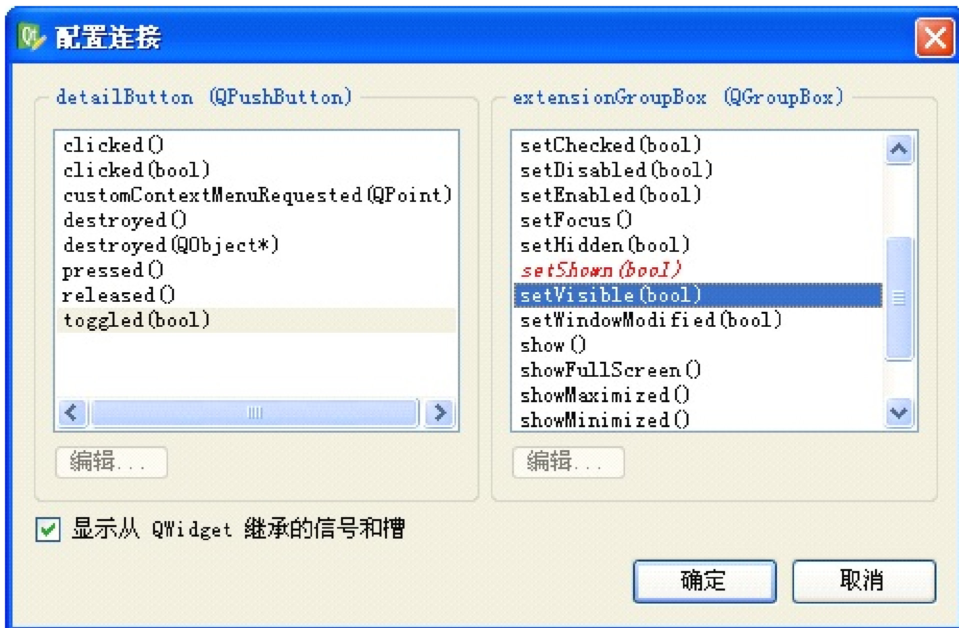


图 7-15 连接按钮的 clicked()信号和窗体的 accept()槽

要建立的第 2 个连接位于【Detail】按钮和【extensionGroupBox】群组框之间。在这 两个窗口部件之间拖动红色箭头线，然后选择 toggled(bool)作为信号，选择 setVisible(bool)作为槽，如图 7-15 所示。

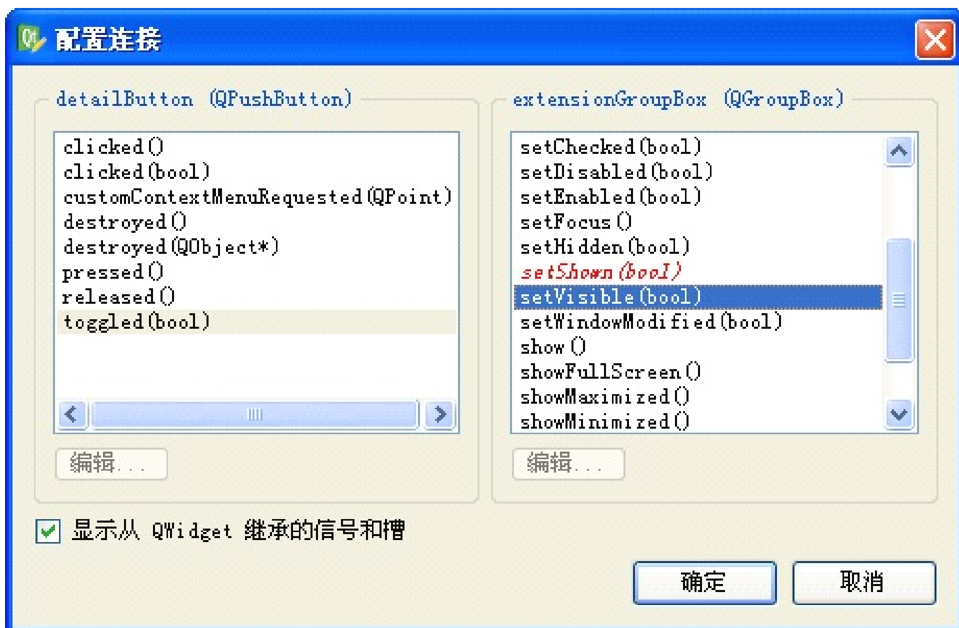


图 7-15 连接按钮的 toggled(bool)槽和分组框的 setVisible(bool)槽

小贴士：默认情况下，setVisible(bool)槽不会显示在 Qt Designer 的槽列表中，但是选 中了【显示从 Qwidget 继承的信号和槽】选项，就可以看到这个槽了。

最后设置完成的信号和槽如图 7-16 所示。

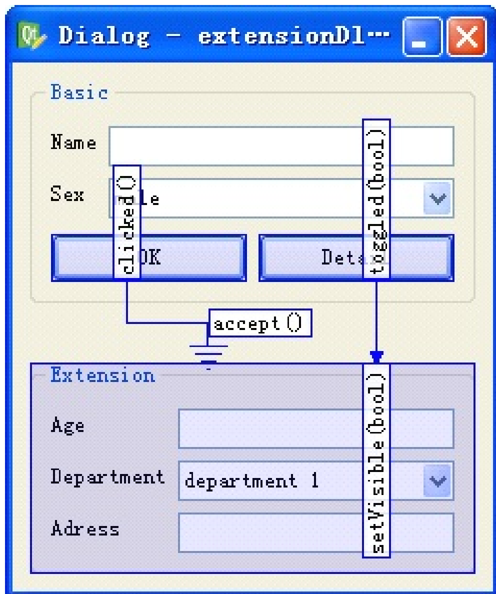


图 7-16 设置完成的信号和槽的情形

第 10 步，保存文件。

将这个对话框保存在前面建立的工程目录下，即 extensionDlgDesigner 目录里面，文 件名为 extensionDlgDesigner.ui。接下来要给这个窗体添加代码，将使用多重继承的方法。

首先用如下内容创建一个 extensionDlg.h 文件：

```
#ifndef EXTENSIONDLG_H
#define EXTENSIONDLG_H
#include &lt;QDialog&gt;
#include "ui_extensionDlg.h"
Class ExtensionDlg:public QDialog,public Ui::Dialog
{
    Q_OBJECT
public:
    ExtensionDlg(QWidget *parent = 0);
};
#endif
```

这里采用的是多重继承的方法，而 Q_OBJECT 宏则是必需的，并且要放在类声明体内最为靠前的地方。

然后再创建 extension.cpp 文件，以下是其内容：

```
#include &lt;QtGui&gt;
#include "extensionDlg.h"
ExtensionDlg::ExtensionDlg(QWidget *parent)
:QDialog(parent)
{
    setupUi(this);
    this-&gt;extensionGroupBox-&gt;hide();
    mainVerticalLayout-&gt;setSizeConstraint(QLayout::SetFixedSize);
}
```

第 1 行调用 setupUi()函数初始化窗体界面元素的布局。

第 2 行表示隐藏了扩展信息窗体，直到用户点击了【Detail】按钮才显示出来。

第 3 行把窗体布局的 sizeConstraint 属性设置为 QLayout::SetFixedSize，这样会使 用户不能再重新修改这个对话框窗体的大小。前面也讲过，这样一来，布局就会负责对话框 重新定义大小的职责，并且也会在显示或者隐藏子窗口部件的时候自动重新定义这个对话框 的大小，从而可以确保对话框总是能以最佳的尺寸显示出来。

最后用下面的代码创建 main()函数，主要就是显示这个对话框。

```
#include "extensionDlg.h"
#include &lt;QApplication&gt;
int main(int argc,char *argv[])
{
    QApplication app(argc,argv);
    ExtensionDlg *dlg = new ExtensionDlg;
    dlg-&gt;show();
    return app.exec();
}
```

添加完这些文件后，可以看看工程文件 extensionDlgDesigner.pro 的内容，这是 Qt Creator 根据程序的情况自动为我们添加好的。

```
HEADERS += extensionDlg.h
SOURCES += extensionDlg.cpp \
main.cpp
FORMS += extensionDlgDesigner.ui
```

程序最后的运行效果如图 7-17 所示，可以看出与手动编写代码的情形是一致的。



图 7-17 程序运行效果

到此，关于扩展对话框的构建就讲解完了。设计一个扩展对话框并不比设计一个简单 对话框困难：所需要的就是一个切换按钮、一些信号 -槽连接以及一个不可以改变尺寸大小 的布局。

小贴士：在实际的应用程序中，控制扩展对话框的按钮通常会在只显示了基本对话框时显 示为 Advanced>>，而在显示了扩展对话框时才显示为 Advanced<<。这在 Qt 中非常容易实现，只需在单击这个按钮时调用 QPushButton 的 setText()函数即可。

通过这个例子，我们讲述了通过子类化构建自定义对话框的方法，以及使用 Qt Designer 辅助设计对话框的方法，在工程实践中，这两种方法往往是结合使用的。

扩展阅读：在 Qt 中，无论是使用手工编码的方式还是使用 Qt Designer，都可以轻松的创建另一种常用的可以改变形状的对话框：多页对话框。可以通过多种不同的方式创建这种对话框：

QTabWidget 的用法就像它自己的名字一样。它提供了一个可以控制内置 QStackedWidget 的 Tab 栏。

QListWidget 和 QStackedWidget 可以一起使用，将 QListWidget::currentRowChanged()信号与 QStackedWidget::setCurrentIndex()槽连接， 然后再利用 QListWidget 的当前项就可以确定应该显示 QStackedWidget 中的哪一页。

与上述 QListWidget 的用法相似，也可以将 QTextWidget 和 QStackedWidget 一起 使用。

在后面的《布局管理》一章中，我们将讲解 QStackedWidget 类。

7.4 常见内建（built in）对话框的使用

内建对话框又被称为是标准对话框。Qt 提供了一整套内置的窗口部件和常用对话框， 如文件选择、字体选择、颜色选择、消息提示对话框等，它们为应用程序提供了与本地平台 一致的观感，可以满足大多数情况下的使用需求。Qt 对这些标准对话框都定义了相应的 类，使用者可以很方便的使用它们。标准对话框在软件设计过程中使经常需要使用的，必须 熟练掌握。

下面我们首先介绍 QInputDialog 、QColorDialog、QFontDialog、、QMessageBox 这 几种标准对话框的使用要领，然后再通过一个实例做示范。

7.4.1 标准输入框（QInputDialog）

本小节讲解如何使用标准输入框。在 Qt 中，构建标准输入框通常使用 QinputDialog 类。QInputDialog 类提供了一种简单方便的对话框来获得用户的单个输入信息。目前 Qt 提 供了 4 种数据类型的输入，可以是一个字符串、一个 int 类型数据、一个 double 类型数据 或者是一个下拉列表框的条目。此外，一般情况下在输入框的附近应该放置一个标签窗口部 件，告诉用户需要输入什么样的值。一个标准输入框的样子如图 7-18 所示。

经常使用的方法有 4 个 : getText()，getInt()， getDouble()和 getItem()，它们都 是 QInputDialog 类的静态方法，使用起来也非常简便，请看下面的示例代码。

```
bool ok;
QString text = QInputDialog::getText(this, tr("User Name"),
    tr("Please input new name"), QLineEdit::Normal,
    QDir::home().dirName(), &ok);
if (ok && !text.isEmpty())
{
    textLabel-&gt;setText(text);
}
```

这段代码将弹出一个对话框请用户输入 "User name"的值，如图 7-18 所示。如果用户按下 OK 按钮，则 ok 的值将为 true，反之将为 false。



图 7-18 标准输入框

7.4.2 标准颜色对话框（QColorDialog）

标准颜色选择对话框被用来为应用程序指定颜色。比如，在一个绘图应用程序中选择 画刷的颜色等。一个典型的标准颜色对话框如图 7-19 所示。

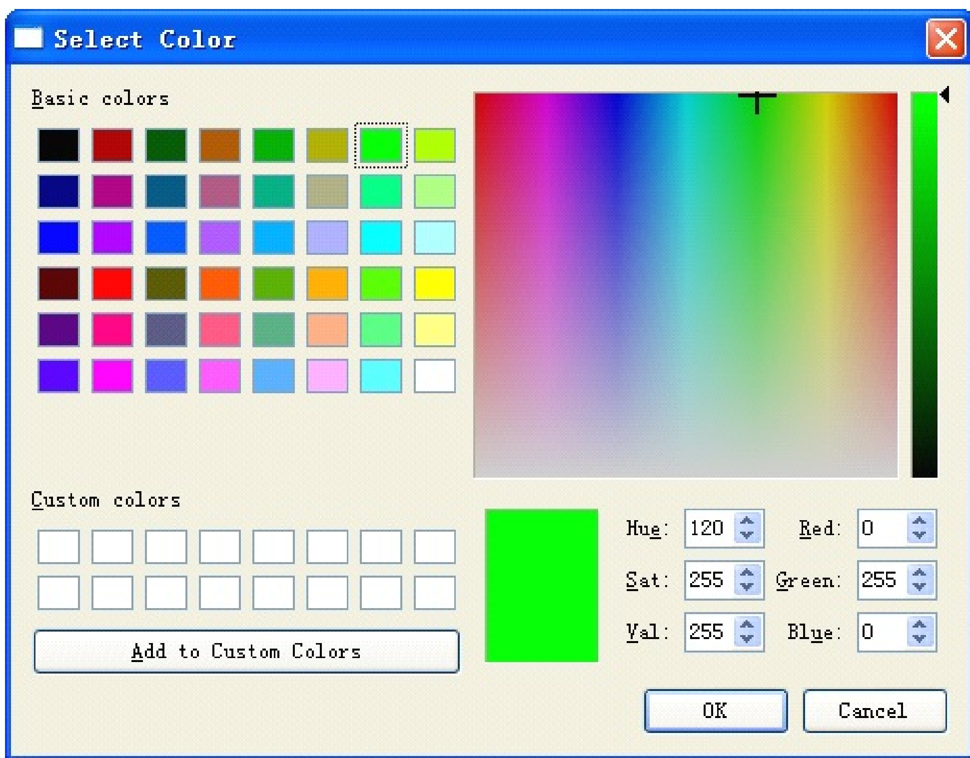


图 7-19 典型的标准颜色对话框

我们经常使用 QColorDialog 类的静态方法 getColor()来创建标准颜色选择对话框。在 其中除了为用户提供了颜色选择面板外，还可以允许用户选择不同的透明度的颜色。

在使用 QColorDialog 类之前，需要引入其头文件声明。

```
#include &lt;QColorDialog&gt;
```

getColor()方法有两种原型，第一种如下：

```
QColor QColorDialog::getColor ( const QColor & initial, QWidget * parent,
    const QString & title,ColorDialogOptions options = 0 ) [static]
```

该方法以 parent 为父窗体，以 initial 为默认颜色，以 title 为窗口标题（如果不指 定的话，将显示为"Select Color"）创建一个模态的颜色对话框，允许用户选择一个字体，并将其返回。如果用户点击了【Cancel】按钮，则返回一个非正常值。这可以通过 bool QColor::isValid () const 方法来校验，如果字体正常则返回值为 ture；反之则返回 false。

另外，通过配置不同的 options 参数能够对颜色对话框的观感进行定制。 options 的取 值来自枚举值 QColorDialog::ColorDialogOption，它也是 Qt4.5 以后引入的，其含义如表 7-2 所示。

表 7-2 枚举值 QColorDialog::ColorDialogOption 的含义

常量	值	说明
QColorDialog::ShowAlphaChannel	0x00000001	允许用户选择颜色的 alpha 值
QColorDialog::NoButtons	0x00000002	不显示 OK 和 Cancel 按钮
QColorDialog::DontUseNativeDialog	0x00000004	使用 Qt 的标准颜色对话框。比如在 Mac OS X 系统中不使用 Apple 的原 生颜色面板

实际中最为常用的是下面这种，它是第一种原型的重载版本。

```
QColor QColorDialog::getColor ( const QColor & initial =
    Qt::white, QWidget * parent = 0 ) [static]
```

一个创建标准颜色对话框的示例代码如下：

```
QColor color = QColorDialog::getColor(Qt::green, this);
if(color.isValid())
{
    colorLabel-&gt;setText(color.name());
    colorLabel-&gt;setPalette(QPalette(color));
    colorLabel-&gt;setAutoFillBackground(true);
}
```

第 1 句创建标准颜色选择对话框。

第 2 句判断颜色是否有效。

第 3 句设置 colorLabel 显示的文本是用户从标准颜色对话框中选择的颜色的名字。

第 4 句通过 setPalette()方法设置 colorLabel 的调色板信息。setPalette()方法经常用于此 种场合。

第 5 句也很重要，setAutoFillBackground()方法用于设置窗体能够自动填充背景。

7.4.3 标准字体对话框（QFontDialog）

标准字体对话框为用户选择字体提供了便捷的途径。 其常见的情形如图 7-20 所示。

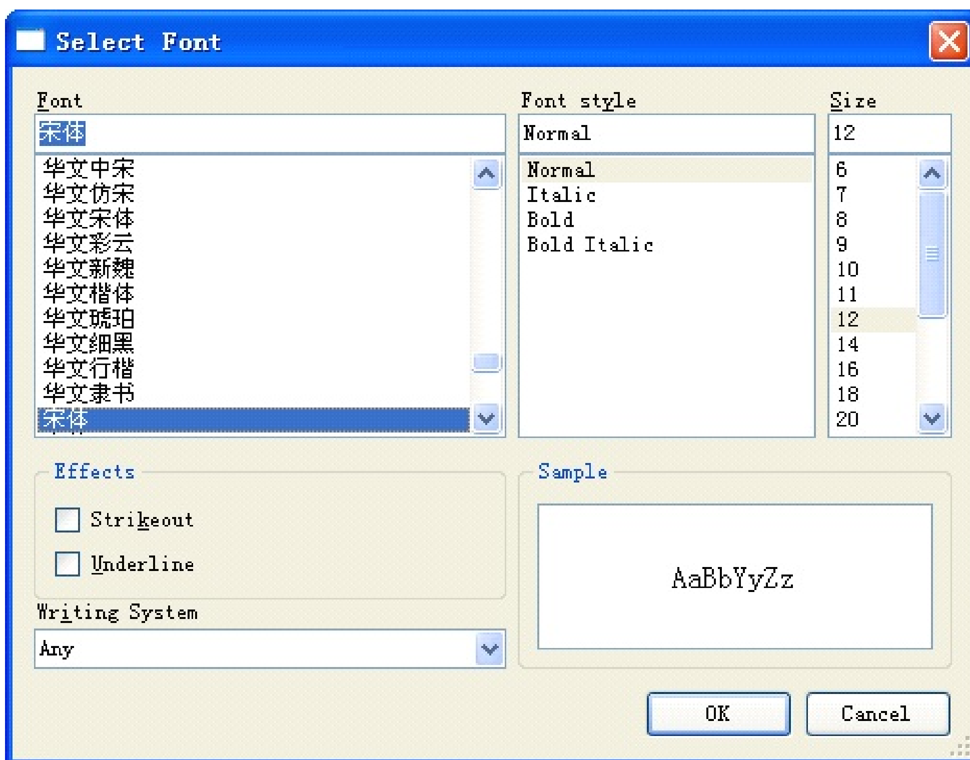


图 7-20 标准字体对话框

通常有两种方式创建字体对话框，一种是使用 QFontDialog 类的构造函数，一种是使用 QFontDialog 类的静态方法 getFont()。在使用 QFontDialog 类之前，应加入其头文件声明：

```
#include &lt;QFontDialog&gt;;
```

1.构造函数第一种原型

```
QFontDialog::QFontDialog ( QWidget * parent = 0 )
```

它将创建一个标准的字体对话框。其中 parent 参数默认即是上下文环境中的父窗口。

这之后可以使用 setCurrentFont()方法来指定初始的字体。该方法是 Qt4.5 以后引进的，一个示例代码如下：

```
QFontDialog fontDlg;
fontDlg.setCurrentFont( QFont(“Times”,12) );
```

2.构造函数第二种原型

```
QFontDialog::QFontDialog ( const QFont & initial, QWidget * parent = 0 )
```

它将使用 parent 作为父窗体，使用 initial 作为默认选择的字体来创建一个标准的字体对话框。该方法是 Qt4.5 以后引进的。一个示例代码如下：

```
QFontDialog fontDlg(QFont(“Times”,12),this);
```

3.使用静态方法 getFont()

它有 6 种原型，最为常用的是

```
QFont QFontDialog::getFont ( bool * ok, const QFont & initial, QWidget * parent,
    const QString & title,FontDialogOptions options ) [static]
```

它创建一个模态的字体选择框，并把用户选择的字体作为返回值。

如果用户点击了【OK】按钮，被用户选择的字体就会被返回。如果用户按下了【Cancel】按钮，那么 initial 将被返回。

它的各个参数的含义是这样的，parent 指定了字体对话框的父窗口，title 是该字体对话 框的标题栏显示的内容，initial 是对话框建立时初始选择的字体。如果 ok 是非空的，当用 户选择了【OK】按钮时，它将被置为 true，当用户选择了【Cancel】按钮时，它将被置为 false。一个示例代码如下：

```
bool ok;
QFont font = QFontDialog::getFont(&ok, QFont("Times", 12), this);
if (ok)
{
    // font is set to the font the user selected
}
else
{
    // the user canceled the dialog; font is set to the initial
    // value, in this case Times, 12.
}
```

这段代码调用 getFont()方法时，对 title 和 options 参数使用了默认值。

也可以使用下面示例的代码直接在某个窗口部件内部使用字体对话框。当用户按下【OK】按钮时，被选中的字体将被使用，当用户按下 【Cancel】按钮时，将仍然使用原来 的字体。

```
myWidget.setFont(QFontDialog::getFont(0, myWidget.font()));
```

小贴士：在字体对话框运行期间，不要删除 parent 指定的父窗口。如果你想这么做的话，就不要使用 getFont()方法来创建字体对话框，而是使用它的构造函数来实现。

下面一种原型也会经常用到，它同样也是 QFontDialog 类的静态方法：

```
QFont QFontDialog::getFont ( bool * ok, QWidget * parent = 0 ) [static]
```

它以 parent 为父窗体创建一个模态的字体对话框，并且返回一个字体。如果用户点击了【OK】按钮，则被选中的字体将被返回，并且 ok 参数将被置成 true；如果用户点击了【Cancel】按钮，则 Qt 默认字体将被返回，ok 参数将被置成 false。

一个创建标准字体对话框的示例代码如下：

```
bool ok;
QFont font = QFontDialog::getFont(&ok, this);
if (ok)
{
    // font is set to the font the user selected
}
else
{
    // the user canceled the dialog; font is set to the default
    // application font, QApplication::font()
}
```

这里同样注意，在使用该方法创建的字体对话框运行期间，不要删除其父窗体。

7.4.4 标准消息对话框（QMessageBox）

在程序开发中，经常会遇到各种各样的消息框来给用户一些提示或提醒， Qt 提供了 QMessageBox 类来实现此项功能。在使用 QMessageBox 类之前，应加入其头文件声明：

```
#include &lt;QMessageBox&gt;
```

Question 消息框、Information 消息框、Warning 消息框和 Critical 消息框的用法大同小异，这些消息框一般都包含一条提示信息、一个图标以及若干个按钮，它们的作用都是给用 户提供一些提醒或一些简单的询问。按图标的不同可以区分为表 7-3 所示的 4 个级别。

	Question	为正常的操作提供一个询问
	Information	为正常的操作报告进行提示
	Warning	报告用户发生了错误，但并不严重
	Critical	报告用户发生了严重的错误

表 7-3 预定义图标的含义

通常有两种方法可以用来创建标准消息对话框。一种是采用 “基于属性的”API，一种是 使用 QMessageBox 的静态方法。这两种方法各有所长，使用静态方法是比较 容易的，但是缺乏灵活性，并且针对用户给出的提示信息不够丰富 ，并且不能自定义消息对话框里面的按钮提示信息。因此，“基于属性的”API 的方法更值得推荐。

1. 基于属性的 API 方法 这种方法的要领如下。

第 1 步，创建一个 QMessageBox 的实例

第 2 步，设置必要的属性信息，通常只需设置 message text 属性即可。 第 3 步，调用 exec()方法显示这个消息框。

下面是一段示例代码。

```
QMessageBox msgBox;
msgBox.setText(tr("The document has been modified."));
msgBox.exec();
```

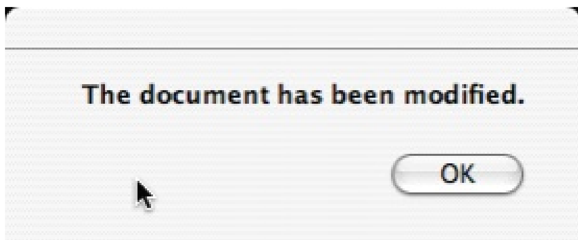


图 7-21 显示一个提示框

这将创建一个提示框，如图 7-21 所示。上面只有一个 OK 按钮，用户必须点击它才能 使该对话框消失，从而结束对话，并且消除模态对话框对交互的阻塞。

2. 使用 QMessageBox 类的静态方法 这种方法使用起来比较简单，一句程序就可以实现，下面 这段示例代码是从下一节中的例子里面来的：

```
QMessageBox::StandardButton reply;
reply = QMessageBox::critical(this, tr("QMessageBox::critical()"),
    MESSAGE,
    QMessageBox::Abort | QMessageBox::Retry | QMessageBox::Ignore);
if (reply == QMessageBox::Abort)
{
    criticalLabel-&gt;setText(tr("Abort"));
}
else if (reply == QMessageBox::Retry)
{
    criticalLabel-&gt;setText(tr("Retry"));
}
else
{
    criticalLabel-&gt;setText(tr("Ignore"));
}
```

第 1 句声明一个 QMessageBox::StandardButton 类型变量，QMessageBox::StandardButton 是一个枚举量，它包含了 QMessageBox 类默认提供的按 钮提示信息，如 OK、Help、Yes、No、Abort、Retry、Ignore 等。

第 2 句调用 QMessageBox 的静态方法 QMessageBox::critical()创建一个 critical 类型 的消息对话框，上面一些提示信息和三个按钮，其效果如图 7-22 所示。

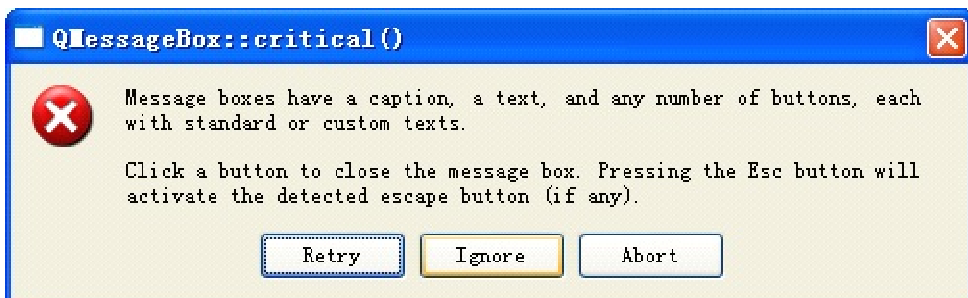


图 7-22 使用静态方法创建标准消息对话框

第 3-8 句则是根据用户选择的按钮不同而把返回值赋给对应的标签部件的文本显示。

7.4.5 实例：标准对话框的使用

下面以一个实例来说明上面这些内置对话框的使用方式和它们之间的区别。本实例主要包含了 7 种类型的消息框，包括 Question 消息框、Information 消息框、Warning 消息框、Critical 消息框、About 消息框、About Qt（关于 Qt）消息框以及 Custom（自定义） 消息框。当用户点击各个按钮时，将创建对应的标准对话框，程序的运行效果如图 7-23 所 示。



图 7-23 综合实例运行效果

我们先来看一下 Dialog 类的头文件。

```
#ifndef DIALOG_H
#define DIALOG_H
#include &lt;QDialog&gt;
QT_BEGIN_NAMESPACE
class QLabel;
class QGroupBox;
class QErrorMessage;
QT_END_NAMESPACE
class Dialog : public QDialog
{
    Q_OBJECT
public:
    Dialog(QWidget *parent = 0);
private slots:
    void getAge();
    void getStature();
    void getSex();
    void getName();
    void getColor();
    void getFont();
    void getCriticalMessage();
    void getInformationMessage();
    void getQuestionMessage();
    void getWarningMessage();
    void getErrorMessage();
    void getAboutMessage();
    void getAboutQtMessage();
    void getCustomMessage();
private:
    QLabel *nameLabel;
    QLabel *sexLabel;
    QLabel *ageLabel;
    QLabel *statureLabel;
    QLabel *colorLabel;
    QLabel *fontLabel;
    QLabel *criticalLabel;
    QLabel *informationLabel;
    QLabel *questionLabel;
    QLabel *warningLabel;
    QLabel *errorLabel;
    QLabel *aboutLabel;
    QLabel *aboutQtLabel;
    QLabel *customLabel;
    QErrorMessage *errorMessageDialog;
};
#endif
```

在文件开始处，首先引入了 QDialog 类的声明，然后前置声明程序中用到的 Qt 类。 Dialog 类公有单继承自 QDialog,接下来为其声明若干个私有槽函数和私有成员变量。

小贴士：类的成员变量尽量声明为私有的。

在 Dialog 类的实现文件的最开始处定义一个宏 MESSAGE，它保存了后面 MessageBox 类的实例用到的提示信息。

```
#include &lt;QtGui&gt;
#include "dialog.h"
#define MESSAGE \
Dialog::tr("&lt;p&gt;Message boxes have a caption, a text, " \
"and any number of buttons, each with standard or custom texts." \
"&lt;p&gt;Click a button to close the message box. Pressing the Esc button " \
"will activate the detected escape button (if any).")
```

然后看一下构造函数。

```
Dialog::Dialog(QWidget *parent)
: QDialog(parent)
{
    //! [1]
    setWindowTitle(tr("Standard Dialogs"));
    errorMessageDialog = new QErrorMessage(this);
    int frameStyle = QFrame::Sunken | QFrame::Panel;
    //! [1]
    //! [2]
    //Name
    nameLabel = new QLabel;
    nameLabel-&gt;setFrameStyle(frameStyle);
    QPushButton *nameButton = new PushButton(tr("&Name(QInputDialog::getText())"));
    //Sex
    sexLabel = new QLabel;
    sexLabel-&gt;setFrameStyle(frameStyle);
    QPushButton *sexButton = new QPushButton(tr("&Sex(QInputDialog::getItem())"));
    //Age
    ageLabel = new QLabel;
    ageLabel-&gt;setFrameStyle(frameStyle);
    QPushButton *ageButton = new pushButton(tr("&Age(QInputDialog::getInteger())"));
    //Stature
    statureLabel = new QLabel;
    statureLabel-&gt;setFrameStyle(frameStyle);
    QPushButton *statureButton = new
    QPushButton(tr("S&tature(QInputDialog::getDouble())"));
    //setup inputdialog groupbox
    QGroupBox *inputDialogGBox = new QGroupBox(tr("Input Dialog"));
    QGridLayout *inputDialogGridLayout = new QGridLayout;
    inputDialogGridLayout-&gt;setColumnStretch(1, 1);
    inputDialogGridLayout-&gt;setColumnMinimumWidth(1, 100);
    inputDialogGridLayout-&gt;addWidget(nameButton, 0, 0);
    inputDialogGridLayout-&gt;addWidget(nameLabel, 0, 1);
    inputDialogGridLayout-&gt;addWidget(sexButton, 1, 0);
    inputDialogGridLayout-&gt;addWidget(sexLabel, 1, 1);
    inputDialogGridLayout-&gt;addWidget(ageButton, 3, 0);
    inputDialogGridLayout-&gt;addWidget(ageLabel, 3, 1);
    inputDialogGridLayout-&gt;addWidget(statureButton, 4, 0);
    inputDialogGridLayout-&gt;addWidget(statureLabel, 4, 1);
    inputDialogGBox-&gt;setLayout(inputDialogGridLayout);
    //! [2]
    //! [3]
    //Color
    colorLabel = new QLabel;
    colorLabel-&gt;setFrameStyle(frameStyle);
    QPushButton *colorButton = new QPushButton(tr("&Color"));
    //Font
    fontLabel = new QLabel;
    fontLabel-&gt;setFrameStyle(frameStyle);
    QPushButton *fontButton = new QPushButton(tr("&Font"));
    //setup color&font Dialog groupbox
    QGroupBox *colorAndFontDlgGBox = new QGroupBox(tr("Color Dialog And Font
    Dialog"));
    QGridLayout *colorAndFontDlgGLayout = new QGridLayout;
    colorAndFontDlgGLayout-&gt;setColumnStretch(1, 1);
    colorAndFontDlgGLayout-&gt;addWidget(colorButton, 0, 0);
    colorAndFontDlgGLayout-&gt;addWidget(colorLabel, 0, 1);
    colorAndFontDlgGLayout-&gt;addWidget(fontButton, 1, 0);
    colorAndFontDlgGLayout-&gt;addWidget(fontLabel, 1, 1);
    colorAndFontDlgGBox-&gt;setLayout(colorAndFontDlgGLayout);
    //! [3]
    //! [4]
    //Question MessageBox
    questionLabel = new QLabel;
    questionLabel-&gt;setFrameStyle(frameStyle);
    QPushButton *questionButton = new QPushButton(tr("&Question"));
    //Information MessageBox
```

```
    informationLabel = new QLabel;
    informationLabel-&gt;setFrameStyle(frameStyle);
    QPushButton *informationButton = new QPushButton(tr("&Information"));
    //Warning MessageBox
    warningLabel = new QLabel;
    warningLabel-&gt;setFrameStyle(frameStyle);
    QPushButton *warningButton = new QPushButton(tr("&Warning"));
    //Critical MessageBox
    criticalLabel = new QLabel;
    criticalLabel-&gt;setFrameStyle(frameStyle);
    QPushButton *criticalButton = new QPushButton(tr("Critica&l"));
    //Error MessageBox
    errorLabel = new QLabel;
    errorLabel-&gt;setFrameStyle(frameStyle);
    QPushButton *errorButton = new QPushButton(tr("Show &Message"));
    //setup messageBox dialog groupbox
    QGroupBox *messageBoxDlgGBox = new QGroupBox(tr("MessageBox Dialog"));
    QGridLayout *messageBoxDlgGLayout = new QGridLayout;
    messageBoxDlgGLayout-&gt;setColumnStretch(1, 1);
    messageBoxDlgGLayout-&gt;addWidget(questionButton, 0, 0);
    messageBoxDlgGLayout-&gt;addWidget(questionLabel, 0, 1);
    messageBoxDlgGLayout-&gt;addWidget(informationButton, 1, 0);
    messageBoxDlgGLayout-&gt;addWidget(informationLabel, 1, 1);
    messageBoxDlgGLayout-&gt;addWidget(warningButton, 2, 0);
    messageBoxDlgGLayout-&gt;addWidget(warningLabel, 2, 1);
    messageBoxDlgGLayout-&gt;addWidget(criticalButton, 3, 0);
    messageBoxDlgGLayout-&gt;addWidget(criticalLabel, 3, 1);
    messageBoxDlgGLayout-&gt;addWidget(errorButton, 4, 0);
    messageBoxDlgGLayout-&gt;addWidget(errorLabel, 4, 1);
    messageBoxDlgGBox-&gt;setLayout(messageBoxDlgGLayout);
    //! [4]
    //! [5]
    //About MessageBox
    aboutLabel = new QLabel;
    aboutLabel-&gt;setFrameStyle(frameStyle);
    QPushButton *aboutButton = new QPushButton(tr("A&bout"));
    //AboutQt MessageBox
    aboutQtLabel = new QLabel;
    aboutQtLabel-&gt;setFrameStyle(frameStyle);
    QPushButton *aboutQtButton = new QPushButton(tr("Ab&out Qt"));
    //custom MessageBox
    customLabel = new QLabel;
    customLabel-&gt;setFrameStyle(frameStyle);
    QPushButton *customButton = new QPushButton(tr("Custom M&essageBox"));
    //setup about dialog groupbox
    QGroupBox *aboutMessageDlgGBox = new QGroupBox(tr("About And Custom MessageBox
    Dialog"));
    QGridLayout *aboutMessageBoxDlgGLayout = new QGridLayout;
    aboutMessageBoxDlgGLayout-&gt;setColumnStretch(1, 1);
    aboutMessageBoxDlgGLayout-&gt;addWidget(aboutButton, 0, 0);
    aboutMessageBoxDlgGLayout-&gt;addWidget(aboutLabel, 0, 1);
    aboutMessageBoxDlgGLayout-&gt;addWidget(aboutQtButton, 1, 0);
    aboutMessageBoxDlgGLayout-&gt;addWidget(aboutQtLabel, 1, 1);
    aboutMessageBoxDlgGLayout-&gt;addWidget(customButton, 2, 0);
    aboutMessageBoxDlgGLayout-&gt;addWidget(customLabel, 2, 1);
    aboutMessageDlgGBox-&gt;setLayout(aboutMessageBoxDlgGLayout);
    //! [5]
    //! [6]
    //connect signals and slots
    connect(ageButton, SIGNAL(clicked()), this, SLOT(getAge()));
    connect(statureButton, SIGNAL(clicked()), this, SLOT(getStature()));
    connect(sexButton, SIGNAL(clicked()), this, SLOT(getSex()));
    connect(nameButton, SIGNAL(clicked()), this, SLOT(getName()));
    connect(colorButton, SIGNAL(clicked()), this, SLOT(getColor()));
    connect(fontButton, SIGNAL(clicked()), this, SLOT(getFont()));
    connect(criticalButton, SIGNAL(clicked()), this, SLOT(getCriticalMessage()));
    connect(informationButton, SIGNAL(clicked()),this,
    SLOT(getInformationMessage()));
    connect(questionButton, SIGNAL(clicked()), this, SLOT(getQuestionMessage()));
    connect(warningButton, SIGNAL(clicked()), this, SLOT(getWarningMessage()));
    connect(errorButton, SIGNAL(clicked()), this, SLOT(getErrorMessage()));
    connect(aboutButton, SIGNAL(clicked()), this, SLOT(getAboutMessage()));
```

```
    connect(aboutQtButton, SIGNAL(clicked()), this, SLOT(getAboutQtMessage()));
    connect(customButton, SIGNAL(clicked()), this, SLOT(getCustomMessage()));
    //! [6]
    //! [7]
    //Setup MainLayout
    QVBoxLayout *mainLayout = new QVBoxLayout;
    mainLayout-&gt;addWidget(inputDialogGBox);
    mainLayout-&gt;addWidget(colorAndFontDlgGBox);
    mainLayout-&gt;addWidget(messageBoxDlgGBox);
    mainLayout-&gt;addWidget(aboutMessageDlgGBox);
    setLayout(mainLayout);
    //! [7]
}
```

在代码块[1]中，首先为应用程序设置标题栏，接着定义了一个 QErrorMessage 的实例，然后定义一个 QFrame 类的实例，用作应用程序窗口部件的风格。关于 QFrame，这里重点讲解一下。

专题：关于 QFrame 类的使用

一般继承自 QWidget 的类都可以设置它的 frame 属性。比如 QLabel 类的窗口在默认情况下拥有一个平面的（flat）观感，而 QProgressBar 类的窗口则拥有一个向下凹陷（sunken）的观感。当然这些都是可以通过设置它们的 frame 而改变的。下面是一段示意代码，演示了如何设置窗口部件的 frame。

```
QLabel label(...);
label.setFrameStyle(QFrame::Panel | QFrame::Raised);
label.setLineWidth(2);
QProgressBar pbar(...);
label.setFrameStyle(QFrame::NoFrame);
```

一个窗口部件的 frame 有两个重要的属性，一个是 shapes，一个是 style。

style 由 QFrame::Shape 和 QFrame::Shadow 枚举值共同来指定，它们可以使设置了 frame 的窗口部件与其他的区分开来。一般可以使用 setFrameStyle()方法来设置 frame 的属性，而使用 frameStyle()方法来读取该设置。

setFrameStyle()的原型如下：

```
void QFrame::setFrameStyle ( int style )
```

它的作用就是设置该窗口部件的 frame 为 style。

常见的各种应用程序的 frame 的观感如图 7-24 所示。

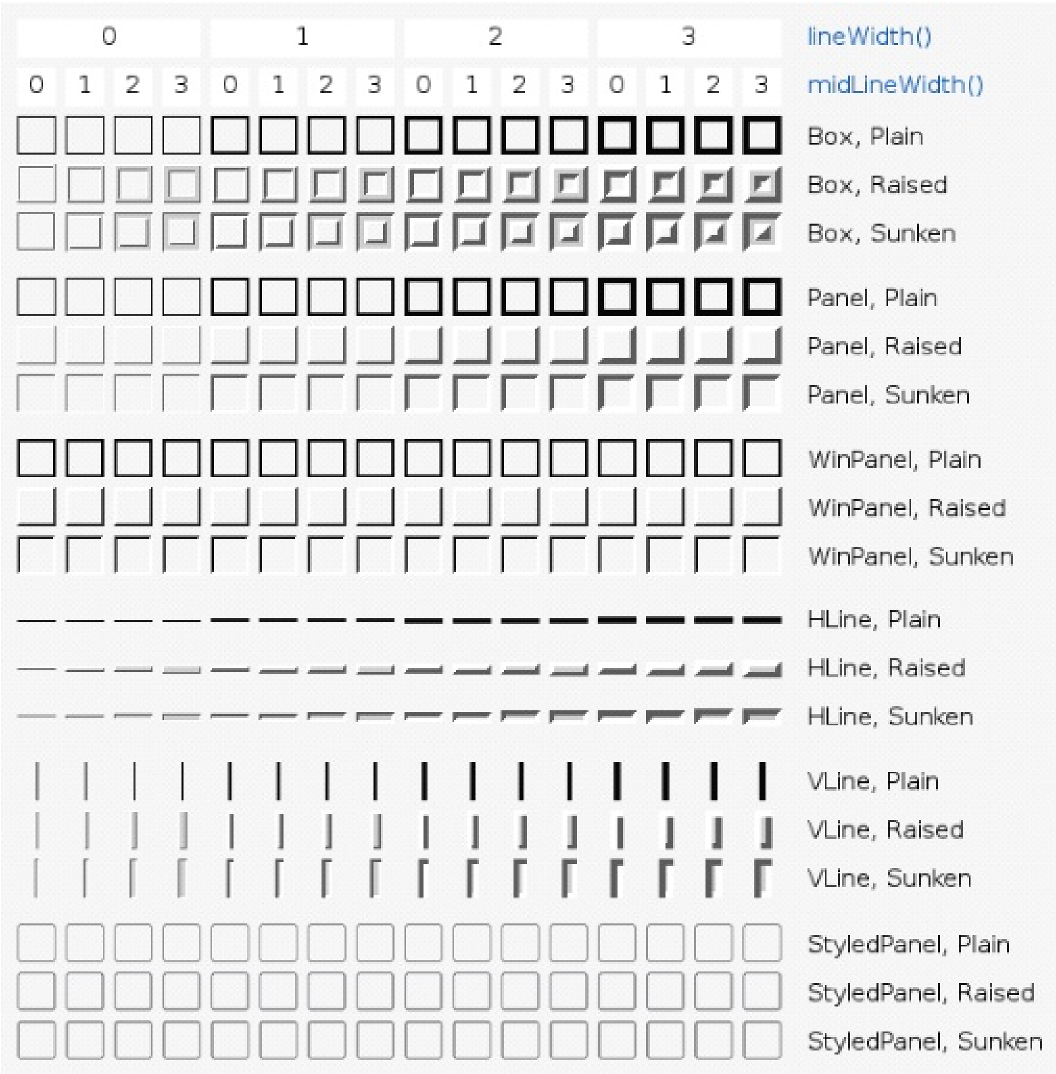


图 7-24 各种 frame 组合起来的效果

在代码块[2]-[5]中，创建了程序中用到的各种窗口部件，然后把它们分类放到几个分 组框中，最后为分组框设置了布局。这其中的重点是 QGroupBox 类的使用，它用作创建分 组框，请看下面的专题讲解。

专题：QGroupBox 类的使用

QGroupBox 为构建分组框提供了支持。分组框通常带有一个边框和一个标题栏，作为容 器部件来使用，在其中可以布置各种窗口部件。分组框的标题通常在上方显示，其位置可以 设置为靠左、居中、靠右、自动调整这几种方式之一。位于分组框之中的窗口部件可以获得 应用程序的焦点。

位于分组框之内的窗口部件被称作是它的子窗口，通常使用 addWidget()方法把子窗口 部件加入到分组框之中。

使用分组框的一般步骤如下：

1. 实例化分组框对象

使用 QGroupBox 构造函数来实例化分组框对象，示例代码如下：

```
QGroupBox *groupBox = new QGroupBox(tr("Exclusive Radio Buttons"));
```

2. 创建位于分组框之中的子窗口部件

同样使用该窗口部件的构造函数来实现，示例代码如下：

```
QRadioButton *radio1 = new QRadioButton(tr("&Radio button 1"));
```

3. 创建一个布局

这个布局就是后面要设置在分组框之上的布局， Qt 提供的常见布局类型比如水平布 局、垂直布局、 栅格布局、表单布局等都可以，目前还不支持分裂器布局。示例代码如下：

```
QVBoxLayout *vbox = new QVBoxLayout;
```

4. 把第 2 步创建的子窗口部件加入到第 3 步创建的布局之中

一般使用 addWidget()或者 insertWidget()方法把子窗口部件加入到布局之中。示例 代码如下：

```
QVBoxLayout *vbox = new QVBoxLayout;
vbox-&gt;addWidget(radio1);
```

5. 把第 3 步创建的布局应用到分组框上

最后，使用 setLayout()方法把布局应用到分组框上，示例代码如下：

```
groupBox-&gt;setLayout(vbox);
```

注意，创建分组框及其内部的子窗口部件时，以上次序不要改变。当分组框内部没有

子窗口部件时，是无法为其应用布局的。再有就是，要牢固记得 QGroupBox 是 QWidget 的子类，QWidget 的公有方法，它的对象都可以使用，它本身就是一个窗口部件。最后，一般情况下，在应用程序中只要使用了分组框，就要为它应用一个布局，请读者朋友注意体会。

下面是一个示例程序片段，它演示了创建分组框并为其设置布局的过程。

```
QGroupBox *groupBox = new QGroupBox(tr("Exclusive Radio Buttons"));
QRadioButton *radio1 = new QRadioButton(tr("&Radio button 1"));
QRadioButton *radio2 = new QRadioButton(tr("R&adio button 2"));
QRadioButton *radio3 = new QRadioButton(tr("Ra&dio button 3"));
radio1-&gt;setChecked(true);
QVBoxLayout *vbox = new QVBoxLayout;
vbox-&gt;addWidget(radio1);
vbox-&gt;addWidget(radio2);
vbox-&gt;addWidget(radio3);
vbox-&gt;addStretch(1);
groupBox-&gt;setLayout(vbox);
```

代码块[6]设置程序中的信号和槽的连接

代码块[7]设置程序总的布局。 接下来看一下设置年龄的函数 getAge()。

```
void Dialog::getAge()
{
bool ok;
int age = QInputDialog::getInteger(this,tr("User Age"),
tr("Please input age"),ageLabel-&gt;text().toInt(),0,150,1,&ok);
if(ok)
{
    ageLabel-&gt;setText(QString(tr("%1")).arg(age));
}
}
```

静态方法 QInputDialog::getInteger 用于创建基于整数的标准输入框。

QString 的 toInt()方法原型如下：

```
int QString::toInt ( bool * ok = 0, int base = 10 ) const
```

它的作用是以 base 为基准将字符串转化为整数。如果 base 等于 0，则表示将使用 C 语 言规定的转换方法，即如果字符串是以 0x 开头的，则转换成 16 进制的整数；如果字符串是以 0 开头的，则转换成 8 进制整数，其它情况则转换成 10 进制整数。一般我们使用十进 制整数的时候较多。

再看一下设置性别的函数 getSex()是如何定义的。

```
void Dialog::getSex()
{
    QStringList items;
    items &lt;&lt; tr("male") &lt;&lt; tr("female");
    bool ok;
    QString sex = QInputDialog::getItem(this,tr("Sex"),
    tr("Please select sex"),items,0,false,&ok);
    if (ok)
    {
        sexLabel-&gt;setText(sex);
    }
}
```

其中，QStringList 类用于容纳一些列的字符串对象。操作符 <<的原型如下：

```
QStringList & QStringList::operator&lt;&lt; ( const QString & str )
```

它的作用于 append()方法相同，将 str 附加到已知的字符串列表中，并返回该列表的引用。

设置字体的函数 getFont()是这样定义的。

```
void Dialog::getFont()
{
bool ok;
QFont font = QFontDialog::getFont(&ok, QFont(fontLabel-&gt;text()), this);
if (ok)
{
    fontLabel-&gt;setText(font.key());
    fontLabel-&gt;setFont(font);
}
}
```

第 3 行调用 getFont()方法创建标准字体对话框。

第 5 行设置 fontLabel 的文本是 font 的文本描述。QFont 的 key()方法的原型如下：

```
QString QFont::key () const
```

它返回该字体的文本描述代码。

第 6 行设置 fontLabel 的字体为用户选中的 font。 现在看看自定义消息对话框的方法，在 getWarningMessage()方法中有很好的示范。

```
void Dialog::getWarningMessage()
{
    QMessageBox msgBox(QMessageBox::Warning, tr("QMessageBox::warning()"),
    MESSAGE, 0, this);
    msgBox.addButton(tr("Save &Again"), QMessageBox::AcceptRole);
    msgBox.addButton(tr("&Continue"), QMessageBox::RejectRole);
    if (msgBox.exec() == QMessageBox::AcceptRole)
    {
        warningLabel-&gt;setText(tr("Save Again"));
    }
    else
    {
        warningLabel-&gt;setText(tr("Continue"));
    }
}
```

首先使用构造函数而不是静态方法创建 QMessageBox 的实例。然后使用 addButton()方法为消息对话框添加按钮。QMessageBox::AcceptRole()和 QMessageBox::RejectRole()是 枚举变量 QMessageBox::ButtonRole 的取值之一，该枚举变量描述了按钮在消息对话框中 的作用。QMessageBox::AcceptRole 表示当用户单击该按钮时，将导致对话框被接受， 类似 于 OK 按钮的作用，而 QMessageBox::RejectRole()表示当用户单击该按钮时，将导致对话 框被拒绝， 类似于 Cancel 按钮的作用。

7.5 模态对话框与非模态对话框

模态对话框（Modal Dialog）与非模态对话框（Modeless Dialog）的概念不是 Qt 所 独有的，在各种不同的平台下都存在。又有叫法是称为模式对话框，无模式对话框等。

所谓模态对话框就是在其没有被关闭之前，用户不能与同一个应用程序的其他窗口进 行交互，直到该对话框关闭。对于非模 态对话框，当被打开时，用户既可选择和该对话框进 行交互，也可以选择同应用程序的其他窗口交互。

在 Qt 中，显示一个对话框一般有两种方式，一种是使用 exec()方法，它总是以模态来 显示对话框；另一种是使用 show()方法，它使得对话框既可以模态显示，也可以非模态显 示，决定它是模态还是非模态的是对话框的 modal 属性。

在 Qt 中，Qt 的模态与非模态对话框选择是通过其属性 modal 来确定的。我们来看看 modal 属性，其定义如下：

```
modal : bool
```

默认情况下，对话框的该属性值是 false，这时通过 show()方法显示的对话框就是非模态的。而如果将该属性值设置为 true，就设置成了模态对话框，其作用于把 QWidget::windowModality 属性设置为 Qt::ApplicationModal。

而使用 exec()方法显示对话框的话，将忽略 modal 属性值的设置并把对话框设置为模 态对话框。

一般使用 setModal()方法来设置对话框的 modal 属性。 我们总结一下设置对话框为模态的方法。

如果要设置为模态对话框，最简单的就是使用 exec()方法，示例代码如下：

```
MyDialog myDlg;
myDlg.exec();
```

也可以使用 show()方法，示例代码如下：

```
MyDialog myDlg;
myDlg.setModal(true);
myDlg.show();
```

如果要设置为非模态对话框，必须使用 show()方法，示例代码如下：

```
MyDialog myDlg;
myDlg.setModal(false);//或者 myDlg.setModal();
myDlg.show();
```

再次强调，目前有的朋友对于模态对话框和非模态对话框的认识有误解，认为使用show()方法显示的就是非模态对话框，这是不正确的。 小贴士：有时候，我们需要一个对话框以非模态的形式显示，但又需要它总在所有窗口的最前面，这时可以通过如下代码设置：

```
MyDialog myDlg;
myDlg.setModal(false);//或者 myDlg.setModal();
myDlg.show();
//关键是下面这行
myDlg.setWindowFlags(Qt::WindowStaysOnTopHint);
```

7.6 问题与解答

问：内建对话框的控件怎么改文本？

比如说 QMessageBox::critical(this, QObject::tr("警告！"), QObject::tr("请正 确输入！"));我要把显示的 OK 按钮文本改为“确定”，怎么获得 OK 的指针？

答：在本章前面已经讲到，这种情况下不要使用 QMessageBox 类的静态方法，下面是 一个类似这种需求时的示例代码：

```
QMessageBox msgBox;
QPushButton *connectButton = msgBox.addButton(tr("Connect"), QMessageBox::ActionRole);
QPushButton *abortButton = msgBox.addButton(QMessageBox::Abort);
msgBox.exec();
if (msgBox.clickedButton() == connectButton)
{
    // connect
}
else if (msgBox.clickedButton() == abortButton)
{
    // abort
}
```

注意重点是 QMessageBox 类的 addButton()方法的使用，其原型可以通过查阅帮助获得。

问：能不能通过对话框的 Title 索引它？

比如我有一个主窗口，其中打开了几个子窗口，我是否可以通过它们的标题来得到它们的指针，继而引用它们？

答：从你的表述来看，你的问题可以归结为如何通过子窗口的 windowTitle 属性来索 引并控制不同的子窗口。

实际上我们很少通过 windowTitle 属性来区分子窗口，因为往往这个属性的值并不是 唯一的，而是可能重复的。所以我们一般是通过 objectName 这个属性来索引不同的子窗 口，继而可以获得其它的属性值，包括 windowTitle。

可以通过 QObject::findChild()、QObject::findChildren()或者是 qFindChild()、qFindChildren()方法来索引子窗口，请看下面的例子，一个 QPushButton 类对象，它的 objectName 属性为"button1"，下面的代码使用 findChild()方法获得了这个子窗口的指针。

```
QPushButton *button = parentWidget-&gt;findChild&lt;QPushButton *&gt;("button1");
```

要获得父窗口的所有子窗口，可以使用 findChildren()方法，下面的代码取得了 parentWidget 的所有类别为 QPushButton 的子窗口：

```
QList&lt;QPushButton *&gt; allPButtons = parentWidget.findChildren&lt;QPushButton *&gt;()
```

需要注意的是 QObject::findChild()、QObject::findChildren()不能与 MSVC6 一起使用，如果要与 MSVC6 配合，则可以换成 qFindChild()、qFindChildren()，它们实现相同的功能。

问：请问我在 Qt Designer 里面设置一个 OKbutton，我将信号 clicked()和槽 accept()相连 接，但为什么运行后点它实现的是关闭对话框的功能呢

答：可以看一下 accept()这个槽的定义：

```
void QDialog::accept () [virtual slot]
//Hides the modal dialog and sets the result code to Accepted.
```

确切说，它是隐藏（hide）这个对话框，看起来的效果是关闭了这个对话框，但是在对话框关闭后，其对象仍然存在，并没有被销毁。 问：如何创建一个不带标题栏的 QMessageBox？ 答：可以使用下面的代码创建，不是使用 QMessageBox 类的静态方法。

```
QMessageBox mgb ;
mgb.setWIndowFlags(Qt::FrameLessTopHint) ;
mgb.exec() ;
```

问：如何让一个对话框总在所有窗口的最前面显示？

答：最简单的方法只需添加一句代码：

```
this-&gt;setWindowFlags(Qt::WindowStaysOnTopHint);
```

问：在 Qt4 中如何为对话框设置背景？

答：这个问题值得总结一下。大致有以下的思路和方法：

1.重写 paintEvent()事件

在该事件中用画笔 QPainter 重画整个背景。该方法比较复杂，不建议初学者采 用。

2.使用调色板类 QPalette

```
//以下代码中 pWidget 为指向要设置的 widget 的指针。
QPalette palette = pWidget-&gt;palette();
palette-&gt;setBrush(QPalette::Active,QPalette::Window,QBrush(Qt::red))
pWidget-&gt;setPalette(palette);
pWidget-&gt;setAutoBackground(true);
```

推荐使用这种方法，大家可以查看一下 setBrush()方法的定义：

```
Palette::setBrush ( ColorGroup group, ColorRole role, const QBrush &brush )
```

这个是一个重载版本，最为重要的是第二个参数，可以设置许多不同的类型。举例来说， 如果 pWidget 指向的是一个 QListWidget 或者 QTextEdit 对象，如果把第二个参数设置 成 QPalette::Text，则是设置其中文字的颜色；如果第二个参数是 QPalette::BrightText，则是设置该对象被选中时，其文字的颜色；如果第二个参数 QPalette::Bright 则表示设置选中该 对象时，高亮背景的颜色或者图片。该函数的功能是很强大的。

3.使用样式表。

```
pWidget-&gt;setStyleSheet("background-color:blue;"); //设置背景颜色 pWidget-&gt;setStyleShe
```



此外，还有一些很有意思的方法，比如用 setHtml(),insetHtml()这种用 HTML 语言的方式来指定的等等，大家可以多查查帮助文档搜索一下 。

4.具体例子

```
//设置为固定颜色
QPalette pal = pWidget-&gt;palette();
pal.setColor(QPalette::Background, QColor(0,0,0)); //黑色
pWidget-&gt;setPalette( pal );
//背景图片
QPalette pal = pWidget-&gt;palette();
pal.setBrush( QPixmap(图片文件名) ); //黑色
pWidget-&gt;setPalette( pal );
```

7.7 总结与提高

对话框为用户提供了许多选项和多种选择，允许用户把选项设置为他们喜欢的变量值并从中做出选择。

本章介绍了如何在 Qt 中创建基于对话框的应用程序。对话框是应用程序中最为常见的 类型之一，因此必须熟练掌握创建对话框的基本方法和技巧，对于基本的概念必须清楚，比 如 QDialog 及其派生类的使用、常见内建对话框的应用以及模态和非模态对话框的区别等 等。对于初学者而言，使用 Qt Creator 作为 IDE，结合 Qt Designer 创建对话框是一种常 见的做法，优点是上手比较快，布局结构可以边修改边预览，但是在工程应用中，比较复杂 的布局通常是采用手工编写代码完成的。所以，笔者还是建议在学习的时候，尽量采用手写 代码子类化 QDialog 的方法完成项目。

第 8 章 主窗口

本章重点

- Qt 中主窗口程序的框架
- 手工创建 QMainWindow 主窗口的基本步骤
- 结合 Qt Designer 创建主窗口的方法
- QAction 动作的创建
- 如何在菜单及工具栏中加入动作
- 如何加入锚接窗口
- 各种锚接方式以及锚接窗体特性的设置方法
- 创建多文档应用程序的方法和步骤

在前面几章中，我们一起学习了 Qt 的基础知识和技能。这一章，我们将带领大家学习 如何创建主窗口应用程序，包括菜单栏、工具栏、状态栏、动作、中心部件、锚接部件等的 创建和使用。最后，将介绍创建多文档窗口的常用方法和步骤。

8.1 主窗口框架

Qt 的 QMainWindow 类提供了一个应用程序主窗口，包括一个菜单栏（ menu bar）、多 个工具栏(tool bars)、多个锚接部件(dock widgets)、一个状态栏(status bar)以及一个 中心部件(central widget)，常见的一种界面布局如图 8-1 所示。



图 8-1 Qt 主窗口常见布局示意图

绝大多数现代 GUI 应用程序都会提供一些菜单、上下文菜单和工具栏。

Qt 通过引入“动作”（action）这一概念来简化有关菜单和工具栏的编程。一个动作 就是一个可以添加到任意数量的菜单和工具栏上的项。

1.菜单栏 菜单是一系列命令的列表。菜单可以让用户浏览应用程序并且处理一些事务，上下文菜单和工具栏则提供了对那些经常使用的功能进行快速访问的方法，它们能够提高软件的使用效率。

为了实现菜单、工具栏按钮、键盘快捷方式等命令的一致性， Qt 使用动作（Action） 来表示这些命令。Qt 的菜单就是由一系列的 QAction 动作对象构成的列表。而菜单栏则是 包容菜单的容器，它通常位于主窗口的顶部，标题栏的下面。一个主窗口通常只有一个菜单 栏。

2.工具栏 工具栏是由一系列的类似于按钮的动作排列而成的面板，它通常由一些经常使用的命令（动作）组成。工具栏的位置处在菜单栏下面、状态栏的上面，工具栏可以停靠在主窗口的上、下、左、右这 4 个不同的位置。一个主窗口可以有多个工具栏。

3.状态栏

状态栏通常是显示 GUI 应用程序的一些状态信息，它位于主窗口的最底部。可以在状 态栏上添加、使用 Qt 窗口部件。一个主窗口只有一个状态栏。

4.锚接部件

对于一个标准的 Qt 主窗口而言，锚接部件不是必需的。锚接部件一般是作为一个容器 来使用，以包容其他窗口部件来实现某些功能。 比如 Qt 设计器的属性编辑器、对象监视器 等都是由锚接部件包容其他的 Qt 窗口部件来实现的。它处在工具栏的内部，可以作为一个 窗口自由的浮动在主窗口的上面，也可以像工具栏一样停靠在主窗口的左、右、上、下四个 方向上。一个主窗口可以包含多个锚接部件。

5.中心窗口部件 中心窗口部件处在锚接部件的内部，它位于主窗口的中心，一个主窗口只有一个中心窗口部件。主窗口 QMainWindow 具有自己的布局管理器，因此在 QMainWindow 窗口上设置布局管理器或者创建一个父窗口部件为 QMainWindow 的布局管理器都是不允许的。但可以 在主窗口的中心窗口部件上设置布局管理器。

6.上下文菜单

为了控制主窗口工具栏和锚接部件的显隐，在默认情况下， QMainWindow 主窗口提供了 一个上下文菜单（Context Menu）。通常，通过在工具栏或锚接部件上单击鼠标右键就可以激活该上下文菜单；也可以通过函数 CMainWindow::createPopupMenu()来激活该菜单。此外，还可以重写 CMainWindow::createPopupMenu()函数，实现自定义的上下文菜单。

8.2 创建主窗口的方法和流程

8.2.1 方法

创建应用程序主窗口界面主要有两种方法：

1.全部代码生成，单继承自 QMainWindow 类，在子类的实现文件中使用代码创建应用 程序主窗口的菜单、工具栏、锚接部件以及状态栏等并设置它们的属性；使用单继承 Qt 窗 口部件类的方法生成中心部件并添加到主窗口中。

2.使用 Qt 设计师绘制应用程序主窗口，在 Qt 设计师中添加菜单（以及子菜单和动 作）、工具栏（以及动作）、锚接部件（以及子窗口部件）、状态栏（目前， Qt 设计师没 有提供状态栏的设计编辑功能，比如无法将窗口部件直接拖放到主窗口的状态栏上）等并设 置它们的属性，以及关联一些基本的信号和槽；然后采用前面介绍的 “单一继承方式”或 “多继承方式”实现应用程序主窗口的代码。这种方法需要和手写代码方法相结合。

一般的，采用第 2 种方法创建应用程序主窗口是比较快的 ，并且具有直观易懂的优 势。

8.2.2 流程

无论采用哪种方法，创建主窗口应用程序一般遵循如下步骤：

1. 创建主菜单

2. 创建子菜单

3. 创建动作

4. 创建工具栏

5. 动作和菜单项以及工具栏按钮的关联

6. 创建锚接窗口（不是必需的）

7. 创建中心窗口部件

8. 创建状态栏

这其中，依据采用手写代码和使用 Qt Designer 的不同，上述步骤有些不是必需的， 或者不是显式的。下面我们先来看看如何使用手写代码创建主窗口程序。

8.3 代码创建主窗口

本实例实现一个基本的主窗口程序，包含一个菜单条、一个工具栏、中央可编辑窗体及状态栏。实现的效果如图 8-2 所示。

8.3.1 头文件

主窗口头文件代码如下：

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H
#include &lt;QMainWindow&gt;
class QAction;
class QMenu;
class QToolBar;
class QTextEdit;
class MainWindow : public QMainWindow  {
    Q_OBJECT
public:
    MainWindow();
    void createMenus();
    void createActions();
    void createToolBars();
    void createStatusBar();
public slots:
    void slotNewFile();
    void slotOpenFile();
    void slotSaveFile();
    void slotCopy();
    void slotCut();
    void slotPaste();
    void slotAbout();
private:
    QTextCodec *codec;
    QMenu *menuFile;
    QMenu *menuEdit;
    QMenu *menuAbout;
    QToolBar *toolBarFile;
    QToolBar *toolBarEdit;
    QAction *actionOpenFile;
    QAction *actionNewFile;
    QAction *actionSaveFile;
    QAction *actionExit;
    QAction *actionCopy;
    QAction *actionCut;
    QAction *actionPaste;
    QAction *actionAboutQt;
    QTextEdit * text;
#endif // MAINWINDOW_H
```

第 1 和第 2 句定义头文件包含卫哨，目的是防止重复包含头文件，这两句与结尾的第 42 句结合在一起使用才是完整的。

第 3 句包含了 QMainWindow 的定义，它是主窗口类的基类。

第 4 至第 7 句对程序下文中可能用到的类进行前置声明（ forward declaration）。它 们会告诉编译器，我们用到的这些类已经存在了，并且不需要知道这些类的完整定义。我们 为什么要这样做，而不是将它们的头文件包含进来呢？这主要是由于在程序下文中，我们只 是简单的定义了指向这些类的对象的指针，而并没有涉及到该类的其他方面。

这样做的好处，一是避免了头文件被其他文件多次包含，尤其是在头文件中包含头文 件时，容易造成重复包含和产生包含顺序问题，并且增大了文件的体积；二是提高了编译速 度，因为编译器只需知道该类已经被定义了，而无需了解定义的细节。

小贴士：尽量不要在头文件中包含另外的头文件 一种好的编程风格是，尽量在头文件中使用类前置声明程序下文中要用到的类，实在需要包含其它的头文件时，可以把它放在我们的类实现文件中。在下面的程序中，你将会看到这个准则的应用。

第 8 句声明了我们的 MainWindow 类是派生自 QMainWindow。

第 10 句的 Q_OBJECT 宏对于所有使用了信号/槽机制的类而言是必需的，同时它要求被 放置在类声明的开始处。

第 12 句声明了我们的主窗口类 MainWindow 的构造函数。

在第 13 至第 16 句中，createActions()函数用于创建程序中用到的动作（Action），createMenus()函数用于创建菜单（Menu），createToolBars()函数用于创建工具栏（ToolBar），CreateStatusBar()函数用于创建状态栏（StatusBar）。接着声明了用到的槽函数，如“新建文件”、“打开文件”等。最后声明了实现主窗口所需的各个元素，包 括菜单项、工具条以及各个动作等。

第 17 至第 24 行，声明了类的槽，这里我们把它们定义为公有的，并且返回值均为 void。需要注意的是，槽同样可以被当做普通函数被调用，这时它的返回值对我们而言与调 用一个普通的 C++函数产生的返回值并无二致。而当槽作为一个信号的响应函数而被执行 时，它的返回值会被程序忽略。也就是说， 不使用信号，我们也可以正常调用槽函数来完 成一些事情。

第 25 至第 40 行，声明了用于实现主窗口所需的各种元素，主要包括 菜单项、工具 条、状态条以及各种动作等，它们是类的成员变量，通常我们被声明为私有的。

8.3.2实现文件

下面我们分析一下主窗口类的实现文件。

1.构造函数

```
#include "mainwindow.h"
#include &lt;QtGui&gt;
// 主窗口实现
CMyMainWindow:: CMyMainWindow()
{
    setWindowTitle(tr("MyMainWindow"));
    text = new QTextEdit(this);
    setCentralWidget(text);
    createActions();
    createMenu();
    createToolBars();
    createStatusBar();
}
```

第 1 行引用主窗口类的头文件。

第 2 行引入 QtGui 模块。

小贴士：在 qmake 工程中，默认情况下已经包含了 QtCore 和 QtGui 模块，因此无需配置就可以使用这两个模块中的类。如果不想使用 QtGui 模块，而仅仅使用 QtCore，就可以在 qmake 工程文件中通过使用”QT -= gui”来取消 QtGui 模块的包含。而对于 Qt 的其他模 块，在使用之前必须在 qmake 工程文件中通过 QT 选项进行配置。在下面的章节中，我们会 有说明。

一般可以在应用程序中通过#include <QtGui/QtGui>来包含整个 QtGui 模块的所有类 的头文件，其中第一个 QtGui 是模块名，第二个 QtGui 是 QtGui 模块（文件夹）下的预定 义头文件；还有一种方法是使用#include <QtGui>，这时 QtGui 表示模块下的预定义头文 件。也可以单独包含某个类的头文件，比如在我们这个主窗口类中引入的 #include <QMainWindow>，或者#include<QtGui/QMainWindow>。

第 5 行设置主窗口的标题，注意我们在程序中并没有使用任何中文的表示方法，因为 这是我们所不提倡的。在这里，你只需知道尽量不要在程序中的任何地方使用汉字，包括注 释，而我们有显示汉字和其他语言形式的方法。但是为了向大家讲解程序方便，我们会在注 释中使用一些汉字。

setWindowTitle 是继承自 QWidget 的方法，它的原型是 void setWindowTitle(const QString &)。再次提醒，要时刻记得 QMainWindow 是继承自 QWidget 的，所以 QWidget 的公有方法，在 QMainWindow 中也有。

第 6 行实例化 QTextEdit 的对象 text，参数 this 指针表示其父窗口为主窗口 CMyMainWindow。

第 7 行设置主窗口的中心窗口部件为 text。在前面我们曾经提到，Qt 主窗口应用程序 中，中心窗口部件是不可或缺的，通常只要是继承自 QWidget 的窗口部件的实例都可以用 作中心窗口部件。在本章的后面我们还会对中心窗口部件做详细的讲解。

第 8、9、10、11 行依次创建动作、主菜单、工具栏和状态栏。

2.创建动作

菜单与工具栏都与 QAction 类密切相关，工具栏上的功能按钮与菜单中的选项条目相 对应，完成相同的功能，使用相同的快捷键与图标。 QAction 类为用户提供了一个统一的命 令接口，无论是从菜单触发还是从工具栏触发，或快捷键触发都调用同样的操作接口，达到 同样的目的。以下是各个动作（Action）的实现代码：

```
void CMyMainWindow::createActions()
{
    // open file action “打开”动作
    actionOpenFile = new QAction(QIcon(":/images/open.png"),tr("Open"),this);
    actionOpenFile-&gt;setShortcut(tr("Ctrl+O"));
    actionOpenFile-&gt;setStatusTip(tr("open a file"));
    connect(actionOpenFile,SIGNAL(triggered()),this,SLOT(slotOpenFile()));
    // new file action “新建”动作
    actionNewFile = new QAction(QIcon(":/images/new.png"),tr("New"),this);
    actionNewFile-&gt;setShortcut(tr("Ctrl+N"));
    actionNewFile-&gt;setStatusTip(tr("new file"));
    connect(actionNewFile,SIGNAL(triggered()),this,SLOT(slotNewFile()));
    // save file action “保存”动作
    actionSaveFile = new QAction(QPixmap(":/images/save.png"),tr("Save"),this);
    actionSaveFile-&gt;setShortcut(tr("Ctrl+S"));
    actionSaveFile-&gt;setStatusTip(tr("save file"));
    connect(actionSaveFile,SIGNAL(activated()),this,SLOT(slotSaveFile()));
    // exit action “退出”动作
    actionExit = new QAction(tr("Exit"), this);
    actionExit-&gt;setShortcut(tr("Ctrl+Q"));
    actionExit-&gt;setStatusTip(tr("exit"));
    connect(actionExit, SIGNAL(triggered()), this, SLOT(close()));
    // cut action “剪切”动作
    actionCut = new QAction(QIcon(":/images/cut.png"), tr("Cut"), this);
    actionCut-&gt;setShortcut(tr("Ctrl+X"));
    actionCut-&gt;setStatusTip(tr("cut to clipboard"));
    connect(actionCut, SIGNAL(triggered()), text, SLOT(cut()));
    // copy action “复制”动作
    actionCopy = new QAction(QIcon(":/images/copy.png"), tr("Copy"), this);
    actionCopy-&gt;setShortcut(tr("Ctrl+C"));
    actionCopy-&gt;setStatusTip(tr("copy to clipboard"));
    connect(actionCopy, SIGNAL(triggered()), text, SLOT(copy()));
    // paste action “粘贴”动作
    actionPaste = new QAction(QIcon(":/images/paste.png"), tr("Paste"), this);
    actionPaste-&gt;setShortcut(tr("Ctrl+V"));
    actionPaste-&gt;setStatusTip(tr("paste clipboard to selection"));
    connect(actionPaste, SIGNAL(triggered()), text, SLOT(paste()));
    // about action “关于”动作
    actionAbout = new QAction(tr("About"), this);
    connect(actionAbout, SIGNAL(triggered()), this, SLOT(slotAbout()));
}
```

第 3～6 行实现的是“打开文件”动作，第 3 行在创建这个动作时，依次指定了此动作 使用的图标、名称以及父窗口。注意程序中 tr()函数的使用很普遍。

小贴士：在程序中需要使用字符串操作时，尽量使用 tr()函数，这是为了日后使得应用程 序可以被翻译为多种语言所必须的。

第 4 行设置了此动作的快捷键为 Ctrl+O，使用的是 setShortcut()方法，它的原型 是：

```
void setShortcut ( const QKeySequence & shortcut )
```

由此引出了使用该方法的另一种方式：

```
actionOpenFile-&gt;setShortcut( QKeySequence( tr("Ctrl+O") ) );
```

专题：QkeySequence 的使用

QKeySequence 是专门用作设置快捷键的。它有 3 种主要的用法：

- 使用 Qt 标准键缩写方式 它的标准写法是：

  ```
  actionOpenFile-&gt;setShortcut( QKeySequence::Open );
  ```

  注意，这种快捷键的预定义是与平台相关的。

- 使用人们已经习惯的类似“Crtl+O”这样的形式 它的标准写法是：

  ```
  actionOpenFile-&gt;setShortcut( QKeySequence( tr("Ctrl+O") ) );
  ```

  它不区分大小写，所以也可写成：

  ```
  actionOpenFile-&gt;setShortcut( QKeySequence( tr("Ctrl+o") ) );
  ```

- 使用在 Qt::Key 和 Qt::Modifier 中定义的前缀修饰符和具体键的组合方式它的标准写法是：

  ```
  actionOpenFile-&gt;setShortcut( QKeySequence( Qt::CTRL + Qt::Key_O ) );
  ```

  其中，Qt::CTRL 表示前置修饰符为 Ctrl 键，Qt::Key_O 表示按下的是字母 O 的键。

Qt::Key 和 Qt::Modifier 是枚举型常量，前者定义了在常见的键值，有上百条之多， 表 8-1 示出了我们这个程序中用到的值，读者可以在 Qt Assistant 中查阅它的所有值；后 者定义了 Qt 所支持的快捷键修饰语的缩写值，目前有 5 种，如表 8-2 所示。

表 8-1 Qt 标准键说明

常量	值	说明
QKeySequence::Close	4	关闭文件
QKeySequence::Copy	9	复制
QKeySequence::Cut	8	剪切
QKeySequence::Delete	7	删除
QKeySequence::HelpContents	1	打开帮助内容
QKeySequence::New	6	创建新文件.
QKeySequence::Open	3	打开文件
QKeySequence::Paste	10	粘贴
QKeySequence::Print	18	打印
QKeySequence::SaveAs	63	另存为
QKeySequence::Save	5	保存文件

表 8-2 Qt::Modifier 中定义的快捷键修饰语值

常量名	值	说明
Qt::SHIFT	Qt::ShiftModifier	在所有标准键盘上都提供
Qt::META	Qt::MetaModifier	Meta 键值.
Qt::CTRL	Qt::ControlModifier	Ctrl 键值.
Qt::ALT	Qt::AltModifier	Alt 键值
Qt::UNICODE_ACCEL	0x00000000	指定 Unicode 值，而不是 Qt 键值

小贴士：在 Mac OS X 中, CTRL 的值 对应 Macintosh 键盘上的命令键， 而 META 的值对应 Windows 平台上的 Ctrl 键值。

在主要使用系统预定义的快捷键的情况下，这是一种比较简便的方法。但快捷键的定 义与平台相关，而且这种写法不利于实现国际化，所以笔者不建议采用。

使用这种方式时，带上 tr()函数便可以支持国际化，并且也与 Windows 平台上大家所 熟悉的写法类似。作者向大家推荐尽量采用这种写法。

最后一种写法看上去比较复杂，并且不利于实现程序的国际化，也不推荐使用。 除了使用 setShortcut 外，定义快捷键还有一种方法，它的标准写法如下：

```
shortcut = new QShortcut(QKeySequence(tr("Ctrl+O", "File|Open")),parent);
```

请读者朋友自行针对我们的程序进行修改，作为一道课后思考题吧。

第 5 行设定了状态条显示，当把鼠标光标移动到此动作对应的菜单条目或工具栏按钮 上时，在状态条上会显示出“打开文件”的提示。

第 6 行连接此动作触发时所调用的槽函数 slotOpenFile()。

3.使用资源文件

首先，大家需要知道在 Qt 工程中，通常使用.qrc 文件来对资源文件进行配置。我们建 议在工程目录下为资源文件建立一个单独的文件夹，以便于管理。在第 8.6 节中，我们会 对资源文件和 Qt 资源系统作专题的讲解。

第 1 步，建立 images 目录

在工程文件夹下面新建一个名为 imgaes 的目录，并将程序工程中需要使用的资源文件（如图标文件、图像文件等）放入该文件夹下面。

第 2 步，建立.qrc 文件

在工程的主目录下面建立一个文本文件，将其保存为 mainwindow.qrc，输入内容如 下：

```
&lt;RCC&gt;
&lt;qresource&gt;
&lt;/qresource&gt;
&lt;/RCC&gt;
&lt;file&gt;images/copy.png&lt;/file&gt;
&lt;file&gt;images/cut.png&lt;/file&gt;
&lt;file&gt;images/new.png&lt;/file&gt;
&lt;file&gt;images/open.png&lt;/file&gt;
&lt;file&gt;images/paste.png&lt;/file&gt;
&lt;file&gt;images/save.png&lt;/file&gt;
```

注意，这是一种 xml 格式的变体，我们需要在成对的标签中放置资源文件。

第 3 步，使工程文件能够识别资源文件 在工程文件（mainwindow.pro）中加入一行：

```
RESOURCES += /mainwindow.qrc
```

这一行是必须的，这样 qmake 才可以找到资源文件。

接下来的各个动作的设置就类似了，“剪切”、“复制”和“粘贴”动作连接的触发 响应槽函数，分别直接使用 QTextEdit 对象的 cut()、copy()和 paste()函数即可。“关 于”动作的触发响应槽函数使用的是 QApplication 的 slotAbout()。

在创建动作时，并不是必须要配合使用图标显示的，例如程序中创建“关于”动作和 “退出”动作时就没有使用图标。这种做法通常用于对应到顶级菜单的动作，并且该顶级菜 单没有子菜单时的情况。

4.创建菜单栏 创建了各个动作后，就可以把它们与菜单栏中的项联系起来了。我们来分析一下 菜单栏的实现函数 createMenus()。

```
void CMyMainWindow::createMenu()
{
    //文件菜单
    menuFile = menuBar()-&gt;addMenu(tr("File"));
    menuFile-&gt;addAction(actionNewfile);
    menuFile-&gt;addAction(actionOpenFileOpen);
    menuFile-&gt;addAction(actionSaveFileSave);
    menuFile-&gt;addAction(actionExit);
    //编辑菜单
    menuEdit = menuBar()-&gt;addMenu(tr("Edit"));
    menuEdit-&gt;addAction(actionCopy);
    menuEdit-&gt;addAction(actionCut);
    menuEdit-&gt;addAction(actionPaste);
    //帮助菜单
    menuAbout = menuBar()-&gt;addMenu(tr("Help"));
    menuAbout-&gt;addAction(actionAbout);
}
```

其中，第 1～5 行创建了“文件”菜单。

第 1 行使用主窗口类的 menuBar()函数得到主窗口的菜单栏指针，再调用菜单栏对象的 addMenu()函数，即可把一个新菜单 menuFile 插入到菜单栏中。

menuBar()函数的原型如下：

```
QMenuBar * QMainWindow::menuBar () const
```

如果菜单栏已经存在，它将返回指向该菜单栏的指针；如果菜单栏还没有建立，它将 创建并且返回一个空的主窗口的菜单栏。

小贴士：menuBar()函数使得该菜单栏以程序中使用的主窗口作为自己的父窗口。因此，如果你想在 Mac 应用程序中使得所有窗口都共用这个菜单栏的话，请不要使用这种写法来创 建菜单栏。要达到这个目的，你可以创建一个没有父窗口的菜单栏，这时我们的代码应该改为：

```
QMenuBar *menuBar = new QMenuBar(0); menuFile = menuBar-&gt;addMenu(tr("File"));
```

另外，创建菜单栏还可以有另一种写法：

```
menuFile = new QMenu(tr("File"),this); QMenuBar *menuBar = menuBar();
menuBar-&gt;addMenu(menuFile);
```

如下面代码所示，addMenu()函数有多个原型，我们通常使用的是前两种。

```
QAction * QMenuBar::addMenu ( QMenu * menu )
QMenu * QMenuBar::addMenu ( const QString & title )
QMenu * QMenuBar::addMenu ( const QIcon & icon, const QString & title )
```

第 2～5 行调用 QMenu 的 addAction()函数在菜单中加入菜单栏条目“打开”、“新建”、“保存”和“退出”。

与上面的情形类似，第 6～9 行创建“编辑”菜单。第 10、11 行创建“帮助”菜单。 5.创建工具栏

接下来创建工具栏，我们来看一下 createToolBars()函数。

```
void CMyMainWindow::createToolBars()
{
toolBarFile = addToolBar(tr("File"));
toolBarFile-&gt;setMovable(false);
toolBarFile-&gt;setAllowedAreas(Qt::AllToolBarAreas);
toolBarFile-&gt;addAction(fileNewAction);
toolBarFile-&gt;addAction(fileOpenAction);
toolBarFile-&gt;addAction(fileSaveAction);
//编辑工具栏
toolBarEdit = addToolBar(tr(“Edit”));
addToolBar( Qt::RightToolBarArea, toolBarEdit);
toolBarEdit-&gt;setMovable(true);
toolBarEdit-&gt;setAllowedAreas( Qt::RightToolBarArea );
toolBarEdit-&gt;setFloatable(true);
QSize size(16, 15);
toolBarEdit-&gt;setIconSize(size);
toolBarEdit-&gt;addAction(copyAction);
toolBarEdit-&gt;addAction(cutAction);
toolBarEdit-&gt;addAction(pasteAction);
//文件工具栏
}
```

第 1～6 行创建了“文件”工具栏，第 5～8 行创建了“编辑”工具栏。

主窗口的工具栏上可以有多个工具条，一般采用一个菜单对应一个工具条的方式，也 可根据需要进行工具条的划分。

第 1 行调用 QMainWindow 的 addToolBar()函数获得主窗口的工具条对象，每新增一个 工具条调用一次 addToolBar()函数，赋予不同的名称，即可在主窗口中新增一个工具条。

专题：创建工具条的方法

addToolBar()函数有 3 种原型，分别介绍如下。

第 1 种：

```
void QMainWindow::addToolBar ( Qt::ToolBarArea area, QToolBar * toolbar )
```

这种方法是最为灵活的一个，第 1 个参数 area 负责设置工具栏的布局方向，比如是从 左到右（left-to-right ），还是从上到下（up-to-down）等等；第 2 个参数 toolbar 是工具栏对象的实例。

举个例子，把我们的程序修改成采用这个函数原型的形式：

第 2 种：

```
void QMainWindow::addToolBar ( QToolBar * toolbar )
```

这种方法是默认将工具栏放置在主窗口的顶部工具栏区域（ Top Toolbar Area ）， 它的作用等同于 addToolBar(Qt::TopToolBarArea, toolbar)，实际上是第 1 种方法的一个具体例子。

这种方法有一个不灵活的地方，就是默认情况下它已经限定工具栏必须创建在主窗口 的顶部工具栏区域，那么由此工具栏的布局方向也已经确定是水平方向的，不可更改了。

第 3 种：

```
QToolBar * QMainWindow::addToolBar ( const QString & title )
```

读者朋友可以看到，我们的程序中就是使用了这种方法。在使用这种方法时，程序会 自动的首先创建一个 QToolBar 对象，并把它的窗口标题设置为 title ，然后将它放置在主窗口的顶部工具栏区域。 使用这种写法，也不能设置工具栏的布局方向。

第 2～4 行调用 QToolBar 的 addAction()函数在工具条中插入属于本工具条的动作。 第 5～8 行编辑工具条的实现，与文件工具条类似。 两个工具条的显示可以由用户进行选择，在工具栏上单击鼠标右键将弹出工具条显示的选择菜单，如图 8-2 所示。

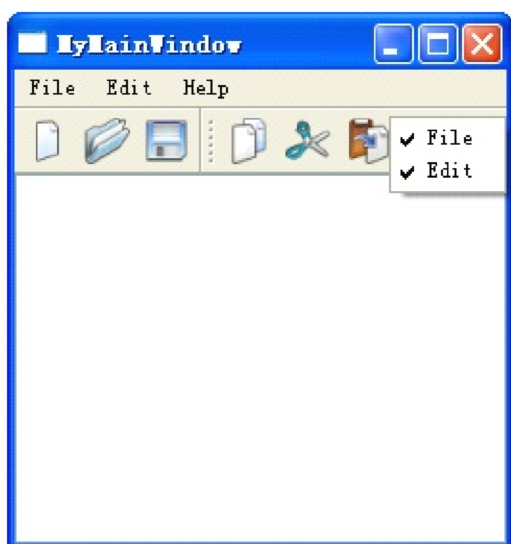


图 8-2 多个工具条可以自由切换

工具栏是可移动和停靠的窗口，它可停靠的区域由 Qt::ToolBarArea 枚举值所决定， 默认值是 Qt::AllToolBarAreas，即出现在主窗口的顶部工具栏区域。

setAllowAreas()函数用来指定工具条可停靠的区域，如：

```
toolBarEdit-&gt;setAllowAreas(Qt::TopToolBarArea | Qt::LeftToolBarArea);
```

它限定了“编辑”工具条只可以出现在主窗口的顶部或左侧。表 8-3 列举了 Qt 工具栏 所有可以停靠的区域。

表 8-3 工具栏可停靠区域- Qt::ToolBarArea 枚举值

常量	值	语义
Qt::LeftToolBarArea	0x1	放置在主窗口左部工具栏区域
Qt::RightToolBarArea	0x2	放置在主窗口右部工具栏区域
Qt::TopToolBarArea	0x4	放置在主窗口顶部工具栏区域
Qt::BottomToolBarArea	0x8	放置在主窗口底部工具栏区域
Qt::AllToolBarAreas	ToolBarArea_Mask	可以放置在主窗口的上、下、左、右 4 个工具栏区域中的任意一个
Qt::NoToolBarArea	0	没有可供工具栏放置的区域

setMovable()函数用于设定工具条的可移动性，如：

```
toolBarEdit-&gt;setMovable(false);
```

该句指定文件工具条不可移动，只出现于主窗口的顶部。

6.创建状态栏

随着菜单和工具栏的完成，已经为设置应用程序的状态栏做好了准备。在程序的普通模式下，状态栏用于显示状态提示和其他的一些临时消息。

```
tipLabel = new QLabel(tr("ready")); tipLabel-&gt;setAlignment(Qt::AlignHCenter);
tipLabel-&gt;setMinimumSize(tipLabel-&gt;sizeHint());
statusBar()-&gt;addWidget(tipLabel);
```

QMainWindow::statusBar()函数返回一个指向状态栏的指针。在第一次调用 statusBar()函数的时候会创建一个状态栏。状态栏指示器一般是一些简单的 QLabel，可以 在任何需要的时候改变它们的文本。当把这些 QLabel 添加到状态栏的时候，它们会自动被 重定义父对象，以便让它们成为状态栏的子对象。

7.实现自定义槽函数 实现新建动作的响应的槽函数 slotNewFile()。

```
void MainWindow::slotNewFile()
{
    MainWindow *newWin = new MainWindow(); newWin-&gt;show();
}
```

新建一个空白文件。必须调用 show()方法使得窗口实例可以显示。

```
void MainWindow::slotOpenFile()
{
    fileName = QfileDialog::getOpenFileName(this); if( !fileName.isEmpty() )
    {
        if( text-&gt;document()-&gt;isEmpty() )
        {

        }
        else
        {

        }
    }
}
loadFile(fileName);
MainWindow *newWin = new MainWindow; newWin-&gt;show();
newWin-&gt;loadFile(fileName);
```

slotOpenFile()槽函数的作用是打开一个文件。利用标准文件对话框 QFileDialog 打 开一个已存在的文件，若当前中央窗体中已有打开的文件，则在一个新的窗口中打开选定的

文件；若当前中央窗体是空白的，则在当前窗体中打开。

具体读取文件内容的工作在 loadFile()函数中完成：

```
void MainWindow::loadFile(QString fileName)
{
    QFile file( fileName );
    if ( file.open( QIODevice::ReadOnly|QIODevice::Text ))
    {
        QTextStream textStream( &file );
        while( !textStream.atEnd() )
        {
            text-&gt;append( textStream.readLine() );
        }
    }
}
```

主要是利用 QFile 和 QTextStream 读取文件内容。

本本例的重点是如何搭建一个基本的 QMainWindow 主窗口，因此对于菜单或工具栏的 具体功能实现并没有做太多的分析，这些功能可在此基本主窗口程序的基础之上逐步完善。

8.4 使用 Qt Designer 创建主窗口

通过对第 5 章的学习，我们已经了解了 Qt Designer 及其组件的基本用法。Qt Designer 提供了许多预定义模板，使用它可以创建多种不同的用户界面。使用 Qt Designer 创建主窗口是非常快捷的，使用其中的 【templates\forms】→【Main Window】就可以完成主窗口类型应用程序界面的设计，主要包括菜单栏、工具栏以及锚接窗口部件的 设定，这其中锚接窗口部件不是必需的。

小贴士：在 Qt4.5 以前的版本中，使用 Qt Designer 不能创建状态栏和锚接窗口部件(Dock Widget)，必须使用手写代码实现；而在 Qt4.5 以后，可以在图形界面下创建锚接窗口部件了，但状态栏还是需要结合手写代码创建。

好了，下面就给出一个完整的创建主窗口应用程序界面的图文流程，示范的环境是在 Windows XP SP2 中文版，Qt4.5.2 开源版下。你在其它操作系统如 Linux 以及 Mac OS X 下的步骤与此类似，只是软件展现的界面风格有所不同。

8.4.1 创建菜单

在一个大型的应用程序中，菜单往往是不可或缺的，它为客户提供了简便快捷的操作模式。菜单有两种：主菜单和上下文菜单。主菜单的位置是固定的，即在应用程序主界面的顶部；上下文菜单一般在用户单击鼠标右键时出现在鼠标的位置，应用更加灵活方便。

第 1 步，选择 Main Window 模板

依次点击 Qt Designer 主菜单上的【文件】→【新建】，弹出如图 8-3 所示的【新建 窗体】对话框，在其中选择【templates\forms】→【Main Window】模板，然后点击【创 建】按钮，即生成一个默认的主窗口界面，如图 8-4 所示。

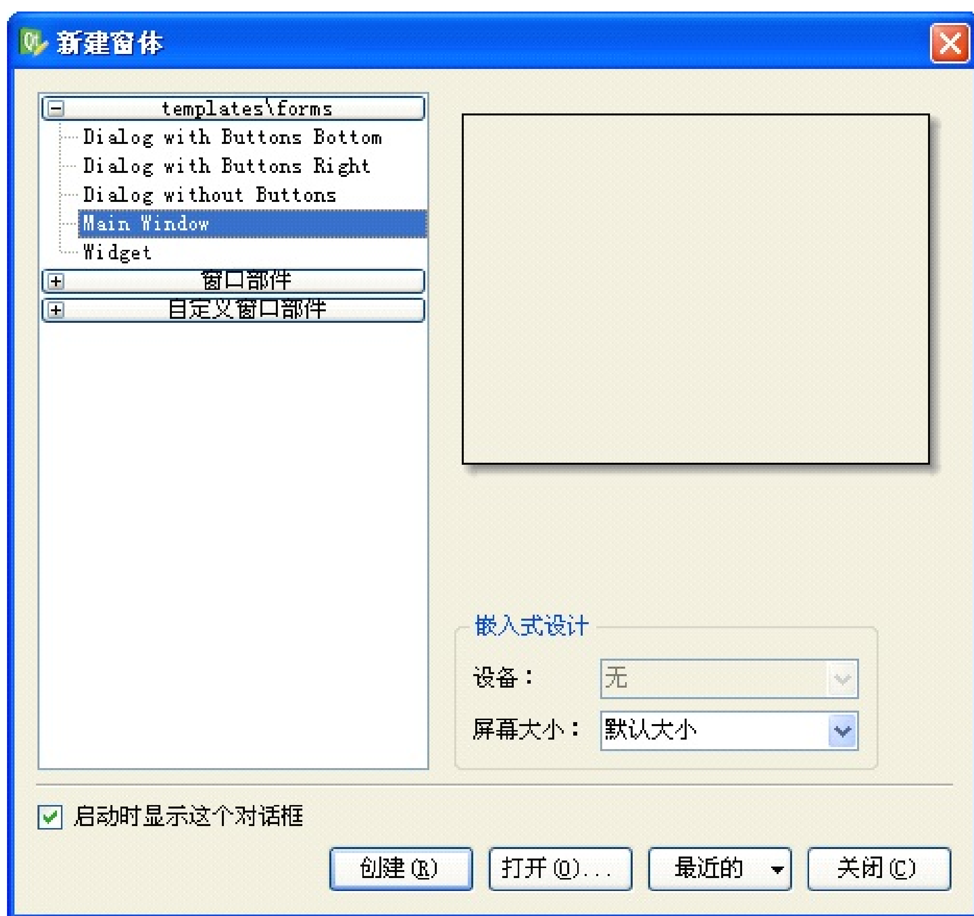


图 8-3 选择 Main Window 模板

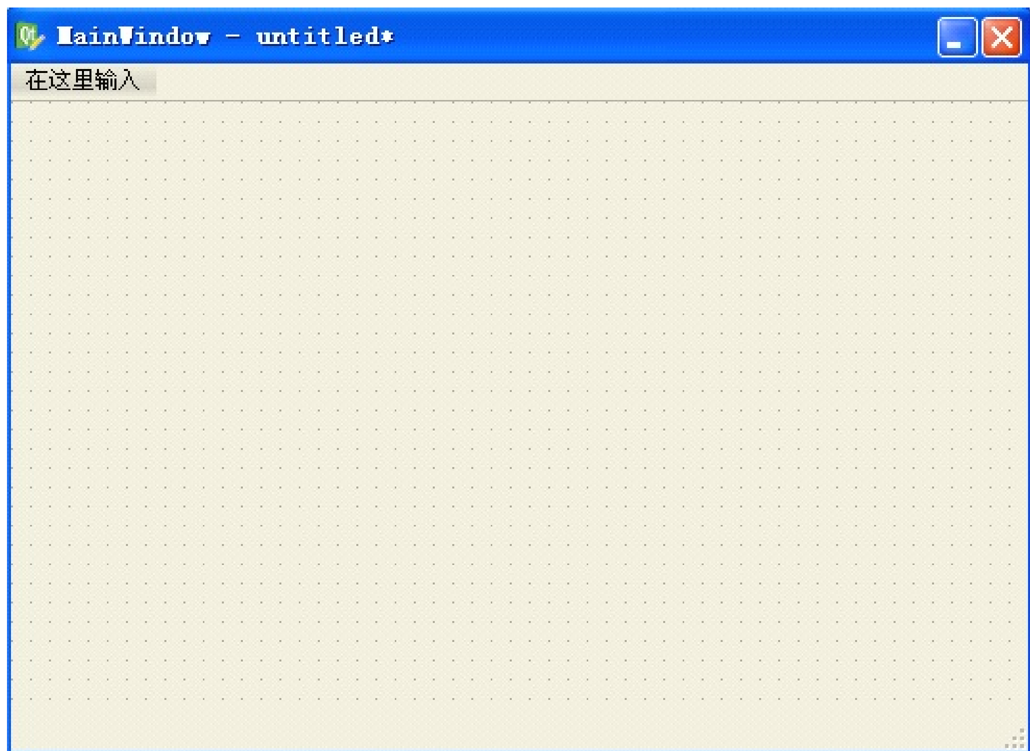


图 8-4 创建了一个空白主窗口界面

在这个系统默认生成的界面上，已经包含了一个菜单栏和一个状态栏，以及一个中心 窗口部件，我们可以从如图 8-5 所示的对象查看器中看得非常清楚。



图 8-5 从对象查看器中看到的界面组成情况

菜单栏是可以被移除的，如图 8-6 所示，方法是在菜单栏上面点击鼠标右键，弹出上 下文菜单，选择【移除菜单栏】即可完成移除；再次创建菜单栏也很容易，在窗体上点击鼠 标右键，在弹出的上下文菜单上选择 【创建菜单栏】命令，即可增加一个新的菜单栏，如图 8-7 所示。状态栏的移除和创建与此相类似，只是状态栏并不可以在界面上编辑，而只能在 属性编辑器中设定属性，在对象观察器中也可设置其名字等基本属性。

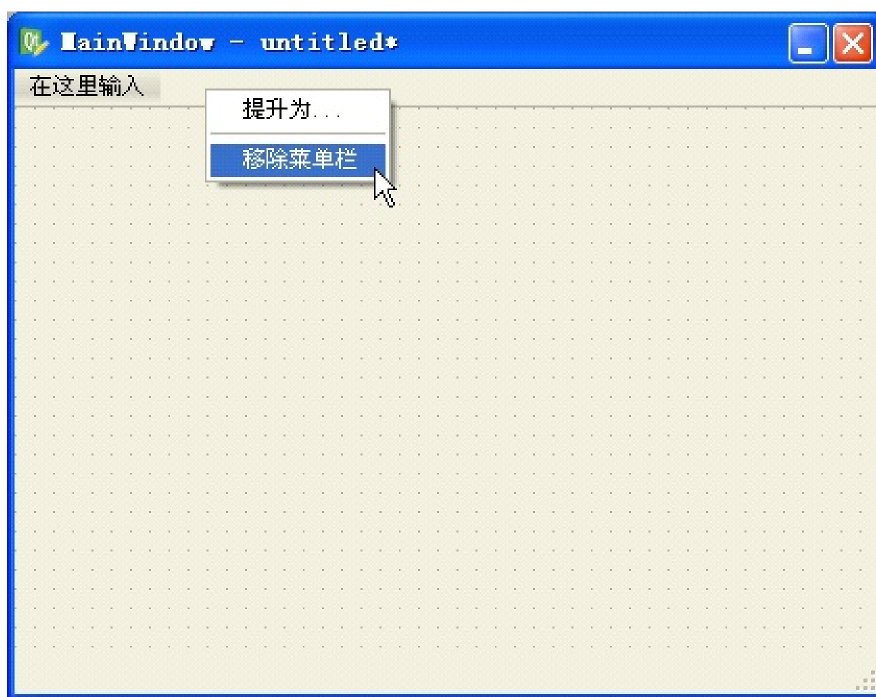


图 8-6 移除菜单栏

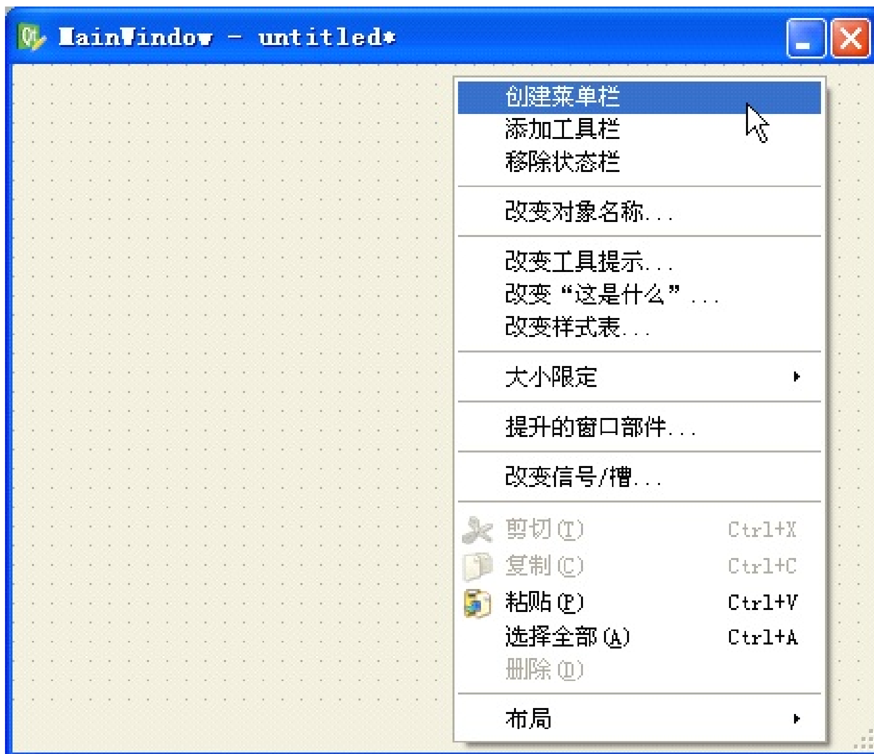


图 8-7 再次创建菜单栏

小贴士：根据 Qt Assistant 中的说法，这个界面默认包含了一个菜单栏和一个工具栏，原文如下：This template provides a main application window containing a menu bar and a toolbar by default -- these can be removed if they are not required. 这是与实际情况不符的，希望读者朋友注意。

一个 Qt 应用程序通常只有一个菜单栏和状态栏，但可以有多个工具栏以及锚接窗口部 件。

第 2 步，创建菜单项。

首先设计主菜单。 在主窗口上用鼠标左键双击【在这里输入】区域，在出现的文本框中输入第一个菜单的名字“&File”，并按下【Enter】键或【Return】键确认输入。其中“&”表示把字符F作为菜单显示文本助记符（在字母 F 下加下划线，即显示为“File”），作用是将 “Alt+F”设置为加速键（accelerator）。当应用程序处在活动状态时，通过按下加速键“Alt+F”，就可以将【File】菜单激活；或者主窗口的菜单栏处在活动的状态下（通过按下”Alt”键就可以将菜单栏激活），在键盘上直接按下 “F”键就可以激活【File】菜 单，如图 8-8 所示。这之后，依次建立【Edit】、【Help】等其它主菜单项。

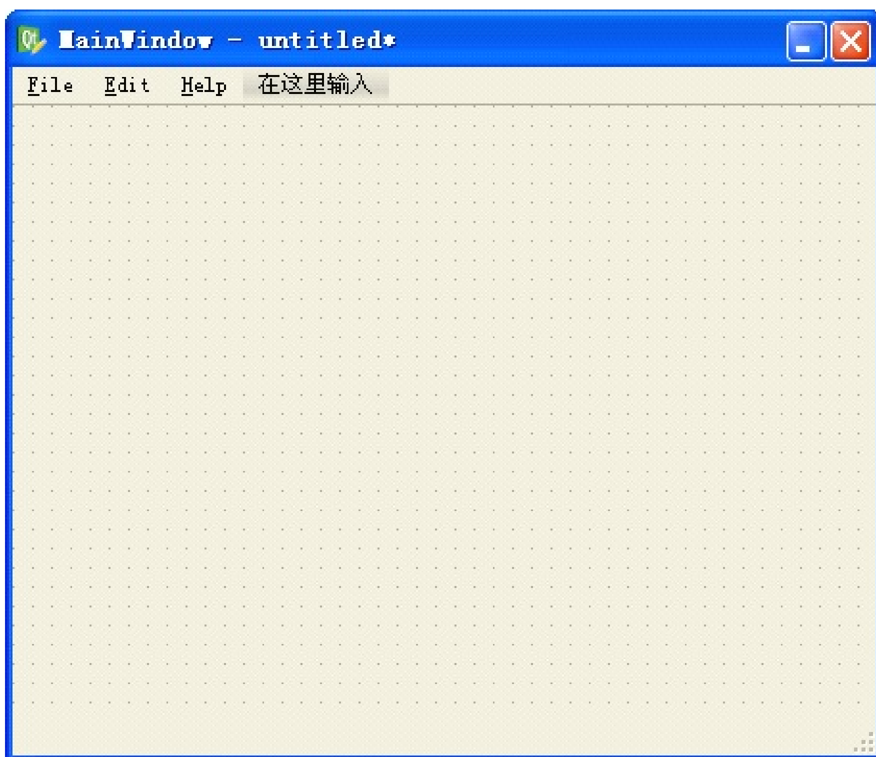


图 8-8 创建主菜单

你也可以在菜单项中输入中文名称，如 “文件”、“编辑”、“帮助”等，但我们不 推荐直接输入中文字符。输入中文的常见做法是 在主窗口上用鼠标左键双击“在这里输 入”区域，在出现的文本框中输入第一个菜单的名字 “文件(&F)”，并按下【Enter】键或【Return】键确认输入。同上，字母 F 将作为助记符，显示为“文件(F)”。其他菜单依此 类推。

小贴士：输入完毕后，一定要按下回车键（Enter 或 Return）予以确认，否则刚输入的字 符将无效。要拒绝输入的字符，可以按下【Escape】键，也可以同时按下“Ctrl+Z”组合键取消刚才的输入。

使用简单的拖拽方式就可以把主菜单项调整到你希望的位置，当你使用鼠标左键拖拽 主菜单项时，一条红色的竖线会提示出目的位置。

主菜单可以有任意多个菜单项，主菜单项的删除可以用上下文菜单命令实现，而不可 以使用【Delete】键。

然后设计子菜单。 子菜单的设计与主菜单类似,需要说明的是如何设置快捷键，这需要借助属性编辑器来完成，在属性编辑器中选中要设定的子菜单项，然后点击 ”shortcut”栏，如图 8-9 所示，在上面按下你想要加入的快捷键，比如 “Ctrl+N”,就同时按下这两个键即可完成设 置。这之后，用同样的做法完成各个子菜单项的添加。



图 8-9 设置子菜单项的快捷键

子菜单也可以有任意多个或多个级联菜单项。子菜单项也可以使用拖拽方式到任意一 个主菜单下面，即使是一个已经关闭的主菜单。要删除子菜单项，可以直接按下 Delete 键。

小贴士：创建子菜单的同时，就创建了动作（ Action），这也是创建动作的方法之一。下面还会讲到如何创建动作，并与菜单项进行关联。

第 3 步，设置属性 设置属性可以做到非常的快捷。请看下面的专题介绍。

专题：对象查看器和属性编辑器的配合使用

首先将对象查看器和属性编辑器按照如图 8-10 所示排列，然后在对象查看器中用鼠标 左键点击选中某一个对象，再切换到属性编辑器里面设置该对象的属性。

这种方法的好处是，不必在界面上费力气的逐个点击对象，再设置属性；而且有些对 象是不能通过鼠标点击获得焦点并设置属性的，如 Status Bar 等。



图 8-10 组合使用对象查看器和属性编辑器

8.4.2创建动作

当菜单栏和工具栏被创建好之后，它们还不能够被使用，必须与动作关联起来。 Qt Designer 提供了一个动作编辑器，使用它创建和管理动作是一件非常轻松的事情。

创建动作有两种做法，一种就是在创建子菜单的同时，一并创建动作；另一种就是先 不创建子菜单，而是使用动作编辑器创建动作，然后再把动作 “拖”到菜单项上，这就在创 建动作的同时也创建了子菜单项。第一种前面已经讲过了，这里重点讲解后一种做法。首先 了解一下动作编辑器的用法。

专题：动作编辑器（Action Editor）

默认情况下，动作编辑器是显示在 Qt Designer 的右半部分的，如图 8-11 所示。

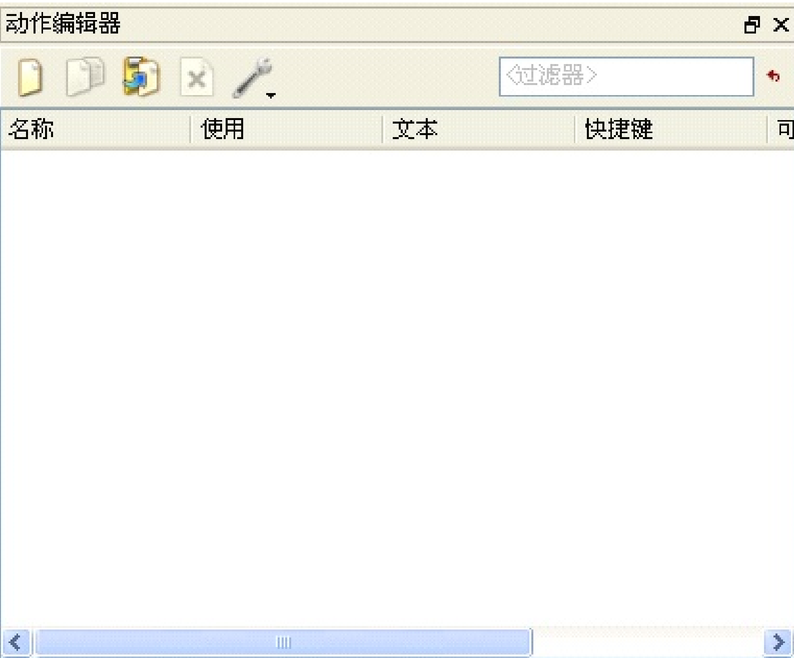


图 8-11 动作编辑器的样子

要控制动作编辑器的显示和隐藏也很容易，方法是依次点击主菜单项 【视图】→【动 作编辑器】，如图 8-12 所示。选中时，该项前面有一个对勾标记，表示显示动作编辑器， 反之，该项前面没有对勾，则表示隐藏动作编辑器。

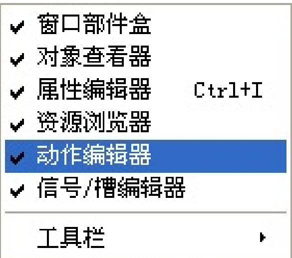


图 8-12 显示或隐藏动作编辑器

动作编辑器允许用户创建、删除动作。它同时也提供了基于动作文本的搜索选项。 动作编辑器可以以图标视图（Icon View）或者是细节视图（Detailed View）的方式浏览。配置的方法是通过鼠标右键的上下文菜单，如图 8-13 所示，通过使用该菜单项可以对 动作进行增加、删除等各种操作。也可以使用动作编辑器工具栏上的快捷按钮。



图 8-13 设置动作编辑器的浏览方式

要创建一个动作，可以在动作编辑器中点击 【新建】按钮，将弹出【新建动作】对话 框，如图 8-14 所示。像图中那样为动作设置各个属性。当一个动作被创建完成后，它就可 以被应用了。

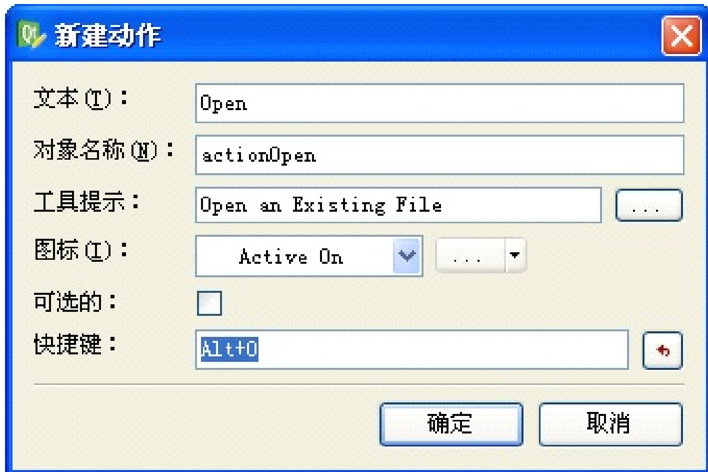


图 8-14 新建一个动作

在配置动作的图标时，如果不满意默认的图标，可以使用资源浏览器创建一个资源文 件（.qrc 文件），并添加各种图标文件。资源浏览器的使用请看后面的专题。

创建一个动作后，还需要把它与菜单或工具栏连接起来。方法是在动作编辑器上的某 个动作上点击鼠标左键，不要松开，把它拖到要关联的菜单或工具栏上。 Qt Designer 提供 了红色的高亮线来指示动作要关联的位置，判断好了之后，就可以释放鼠标左键，从而把动 作与菜单或工具栏按钮关联起来。图 8-15 显示了设置动作与菜单项关联的情形，图 8-16 显示了设置动作与工具栏按钮关联的情形。



图 8-15 设置动作与菜单项的关联



图 8-16 设置动作与工具栏按钮的关联

8.4.3 创建工具栏

创建工具栏很容易，通过使用鼠标右键的上下文菜单项 【添加工具栏】即可，如图所 示。同样的，如果已经有了工具栏，同样可以通过该上下文菜单项移除工具栏。



图 8-17 添加工具栏

要创建工具栏上的按钮，就必须在动作已经被创建的情况下才能进行。一个比较好的 方法是从动作编辑器中拖动动作至工具栏上，在认为合适的地方松开鼠标左键，这时就创建 了一个工具栏按钮。具体做法与上一小节介绍的相同。

工具栏上的按钮是可以被分组的，方法是使用鼠标右键的上下文菜单项 【添加分隔 符】，这样为【File】和【Edit】菜单分别创建工具组，如图 8-18 所示。

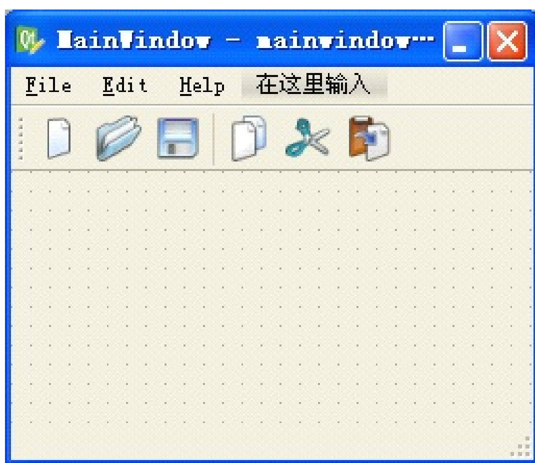


图 8-18 设置好工具栏的主窗口程序

8.4.4 创建锚接窗口部件（不是必需的）

从 Qt 4.5 开始，Qt Designer 在其窗口部件盒中可视化的提供了锚接窗口部件，这就 进一步简化了创建锚接窗口的步骤。因此，为主窗口程序创建锚接窗口比较简单，就是把这个部件从窗口部件盒中拖出来，放到合适的位置。如图 8-19 所示的例子中，一共放置了 4个锚接窗口部件，而且上、下、左、右这 4 个可供停靠的位置上可以放置不止一个锚接窗 口部件。

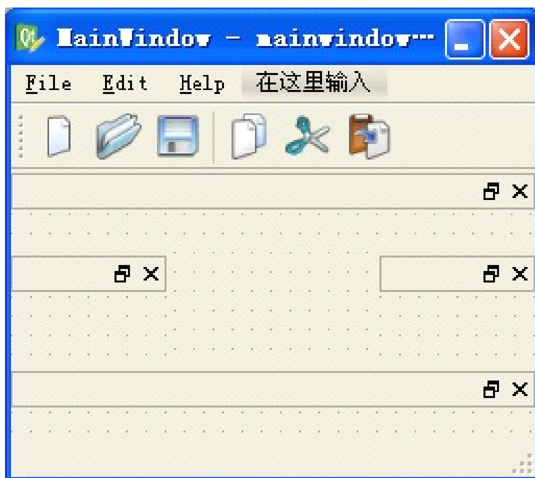


图 8-19 添加锚接窗口部件

注意，拖动锚接窗口部件时，不必太在意它的初始位置，这可以在属性编辑器中调 整，其锚接位置主要是通过 dockWidgetArea 属性来设置，如图 8-20 所示。

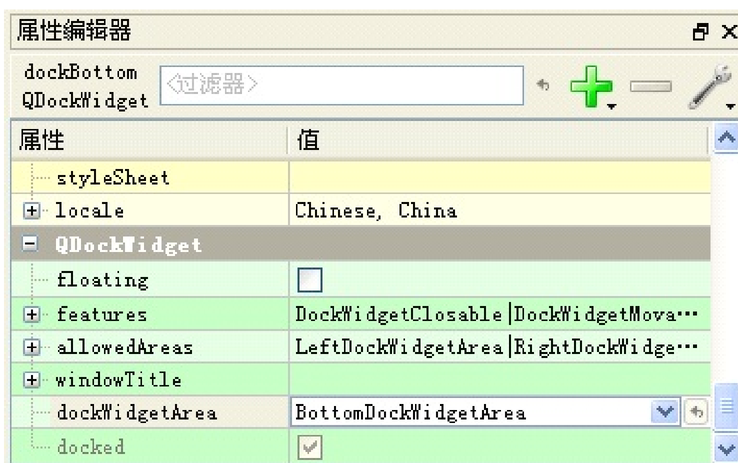


图 8-20 设置 dockWidgetArea 属性

8.4.5 创建中心窗口部件

在 Qt Designer 中为主窗口创建中心窗口部件，只需要将一个窗口部件拖动到窗口部 件盒中，然后再设置一个布局即可， Qt 会为程序设置好中心部件。如图 8-21 所示，图中 放置了一个 TextEdit 窗口部件，然后设置了一个垂直布局。

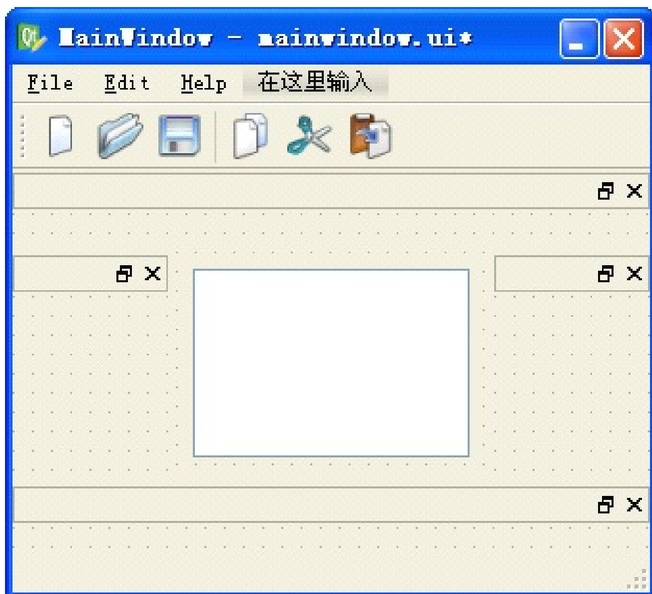


图 8-21 设置好中心窗口部件

我们可以从对象查看器中看到布局以及中心窗口部件的设置情况，如图 8-22 所示。

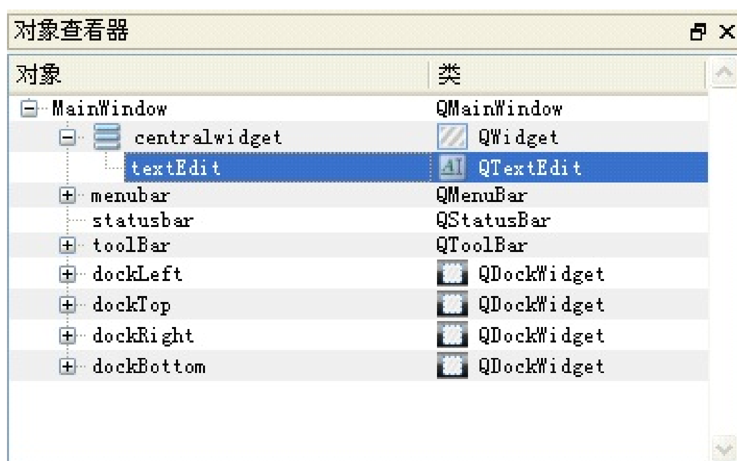


图 8-22 布局以及中心窗口部件的情况

好了，到此我们就使用 Qt Designer 创建了一个完整的主窗口界面，步骤也是比较清 晰的。中间用到了属性编辑器、对象浏览器、资源浏览器和动作编辑器以及窗口部件盒等重 要的工具，几乎将 Qt Designer 所用到的功能练习了一遍。使用 Qt Designer 创建主窗口 界面后，仍然需要使用代码完成菜单项的功能，可以采用手写代码的方式，也可以借助于 Qt Creator 这样的 IDE。请读者朋友自行练习。

8.5 中心窗口部件专题

从前面的例子中我们可以看到，Qt 程序中的主窗口通常具有一个中心窗口部件。从理 论上来讲，任何继承自 QWidget 的类的派生类的实例，都可以作为中心窗口部件使用。

8.5.1 几种常见情形

QMainWindow 的中心区域可以被任意种类的窗口部件所占用。下面给出的是所有可能情 形的概述。

1. 使用标准的 Qt 窗口部件（Standard Widget）

像 QWidget、Qlabel 以及 QTextEdit 等等这样的标准窗口部件都可以用作中心窗口部 件。

2. 使用自定义窗口部件（User-Define-Widget）

有时候，某些有特殊要求的应用程序需要在自定义窗口部件中显示数据，你可以把自 定义的窗口部件作为中心窗口部件。例如，你的绘图编辑器程序程序就可以使用类似名为 PhotoEditor的自定义窗口部件作为自己的中心窗口部件。

3. 使用一个带布局管理器的普通 Widget 有时候，应用程序的中央区域会被许多窗口部件所占用。这时就可以通过使用一个作为所有这些其他窗口部件父对象的 QWidget，以及通过使用布局管理器管理这些子窗口部件的大小和位置来完成这一特殊情况。

4. 使用切分窗口（QSplitter） 其实，这种情况是上一种情况的一个例子。多个窗口部件一起使用的另一种方法是使用 QSplitter。我们把 QSplitter 作为一个容器，在其中容纳其它的窗口部件，这时的中心窗口部件就是一个 QSplitter。QSplitter 会在水平方向或者竖直方向上排列它的子窗口部 件，用户可以利用切分条（splitter handle）控制他们的尺寸大小。切分窗口可以包含所 有类型的窗口部件，包括其他切分窗口。

5. 使用多文档界面工作空间（QMdiArea） 如果应用程序使用的是多文档界面，那么它的中心区域就会被 QMdiArea 窗口部件所占据，并且每个多文档界面窗口都是它的一个子窗口界面。 QMdiArea 是在 Qt4.3 以后引入的一个支持多文档应用的类。

6. 使用工作空间部件（QWorkspace） 这种情况通常用于多文档应用程序中，这时应用程序主窗口的中心部件是一个 QWorkspace 部件或者它的子类化部件。但这种方法在 Qt4.5 以后将被废弃。后面我们还会讲到它。

8.5.2 创建和使用

一个 Qt 主窗口应用程序必须有一个中心窗口部件 (Central Widget)。当你采用 Qt Designer 创建主窗口时，默认情况下，系统已经为你创建了一个中心窗口部件，它是子类 化 QWidget 的。

结合代码可以方便的设置中心窗口部件，可以调用主窗口类的 setCentralWidget()方 法，它的原型如下：

```
void QMainWindow::setCentralWidget ( QWidget * widget )
```

它将把 widget 设置为主窗口的中心窗口部件。

创建中心窗口部件完整的代码示例如下：

```
QTextEdit * text;
text = new QTextEdit(this); setCentralWidget(text);
```

Qt 应用程序的主窗口管理着中心窗口部件，它会在合适的时候销毁这个中心窗口部 件。每次程序调用 setCentralWidget()方法时，先前存在的中心窗口部件将被新的所替 换，而且主窗口会销毁原来的部件，无需用户处理。

要想在程序中获得并使用、设置中心窗口部件也很简单，通过调用主窗口类的 centralWidget()方法即可实现，它的函数原型如下：

```
QWidget * QMainWindow::centralWidget () const
```

它将返回主窗口的中心窗口部件，如果中心窗口部件不存在，它将返回 0。一个完整的 示例代码如下：

```
QTextEdit * text = centralWidget();
//设置属性，但通常不需要
...
```

8.6 Qt4 资源系统专题

Qt4 资源系统是与平台无关的，它被用来存储应用程序可执行文件运行时使用的二进制 文件（比如图标文件、翻译文件等）。它也是 Qt 的核心机制之一。当你的应用程序总是使 用一些特定的文件集合时，它会非常有用，并且能够保证文件不易丢失。

Qt4 资源系统的运转需要 qmake、rcc（(Qt's resource compiler）以及 QFile 的紧密 配合。

8.6.1 Qt 资源系统的改进

Qt4 的资源系统取代了 Qt3 的 qembed 工具和图片集（image collection）机制。 仍以我们的主窗口程序为例，如果采用 Qt3 的图片集机制，那么需要将下述代码加入到工程文件 mainwindow.pro 中：

```
IMAGES = images/icon.png \
images/open.png \
......
images/find.png \ images/gotocell.png \
images/new.png \
```

而使用 Qt4 的资源系统机制的话，只需要在 mainwindow.pro 中加入一条代码：

```
RESOURCE = mainwindow.qrc
```

从中我们可以看出，Qt3 的图片集机制是把资源文件的分布情况罗列出来，逐条的写在 工程文件.pro 中，当程序中经常用到的资源文件数量非常多时，书写工程文件将是一件颇费力气而且乏味的工作，而且这样也会导致 .pro 文件条理不清晰，维护困难，容易出错。

8.6.2 Qt4 资源集文件

Qt4 在 Qt3 的基础上做了改进。重新定义了一个 Qt 资源集（Resource Collection Files）文件，即.qrc 文件，它是一个基于标准 xml 格式的文本文件，我们在程序中所用到 的资源文件就被有规律的嵌入到它的标签中，条理非常清晰，易于阅读和维护。 Qt4 的 qmake 能够识别这个资源集文件，并且能够根据它的描述去相应的目录下定位具体的资源。

8.6.3 资源文件的使用方法

在构造函数的最后部分，把窗口的图标设置为 icon.png，它是一个 PNG 格式的文件。 Qt 支持很多图像格式，包括 BMP、GIF、JPEG、PNG、PNM、SVG、TIFF、XBM 和 XPM。调用 QWidget::setWindowsIcon()函数可以设置显示在窗口左上角的图标。遗憾的是，还没有一种与平台无关的可在桌面上显示应用程序图标的设置方法。

图形用户界面（GUI）应用程序通常会使用很多图片。为应用程序提供图片的方法有多 种，以下是最为常用的一些方法：

1. 把图片保存到文件中，并且在程序运行时载入它们。

2. 把 XPM 文件包含在源代码中。（这一方法之所以可行，是因为 XPM 文件也是有效的 C++文件。）

3. 使用 Qt 的资源机制（Resource Mechanism）。

这里，我们使用了 Qt 的资源机制法，因为它比运行时载入文件的方法更为方便，并且 该方法适用于所支持的任意文件格式。我们将选中的图片存放在源代码树中名为 images 的 子目录下。

为了利用 Qt 的资源系统，必须创建一个资源文件（ Resource File），并且在识别该 资源文件的.pro 工程文件中添加一行代码。在这个例子中，已经将资源文件命名为 xxx.qrc，因此只需在.pro 文件中添加如下一行代码：

```
RESOURCE = xxx.qrc
```

资源文件自身使用了一种简单的 xml 文件格式。这里给出的是从已经使用的资源文件 中摘录的部分内容：

```
&lt;RCC&gt;
&lt;qresource&gt;
&lt;file&gt;images/copy.png&lt;/file&gt;
&lt;file&gt;images/cut.png&lt;/file&gt;
&lt;file&gt;images/new.png&lt;/file&gt;
&lt;file&gt;images/open.png&lt;/file&gt;
&lt;/qresource&gt;
&lt;/RCC&gt;
&lt;file&gt;images/paste.png&lt;/file&gt;
&lt;file&gt;images/save.png&lt;/file&gt;
```

所有资源文件都会被编译到应用程序的可执行文件中，因此并不会弄丢它们。当引用 这些资源时，需要使用带路径前缀 :/（冒号斜线）的形式，这就是为什么会将图标文件表示成:/images/icon.png 的形式。资源可以使任意类型的文件（并非只是一些图像），并且可以在 Qt 需要文件名的大多数地方使用它们。

通过 Qt 资源编辑器 rcc，可以将资源转换为 C++代码。还可以通过把下面一行代码加 入到.pro 文件中来告诉 qmake 包括专门的规则以运行 rcc。

```
RESOURCE = myresourcefile.qrc
```

myresourcefile.qrc 文件是一个 XML 文件，它列出了所有嵌入到可执行文件中的文 件。

假设我们正在编写一个保持联系细节的应用程序。考虑到用户使用的方便性，我们想 在最后的可执行文件中嵌入国际拨号代码。如果文件在应用程序所建目录的 datafiles 目 录下，那么资源文件将会如下所示：

```
&lt;RCC&gt;
&lt;qresource&gt;
&lt;file&gt;datafiles/phone-codes.dat&lt;/file&gt;
&lt;/qresource&gt;
&lt;/RCC&gt;
```

在应用程序中，资源是通过:/路径前缀识别的。在这个例子中，拨号代码文件的路径为:/datafiles/phone-codes.dat，它可以像其他任何文件一样通过 QFile 读取。

在可执行文件中的嵌入数据具有不易丢失的优点，而且也有利于创建真正独立的可执 行文件（如果也采用了静态链接的话）。它的两个缺点是：第一，如果需要改变嵌入数据， 则整个可执行文件都要跟着替换；第二，由于必须容纳被嵌入的数据，可执行文件本身将变 得比较大。

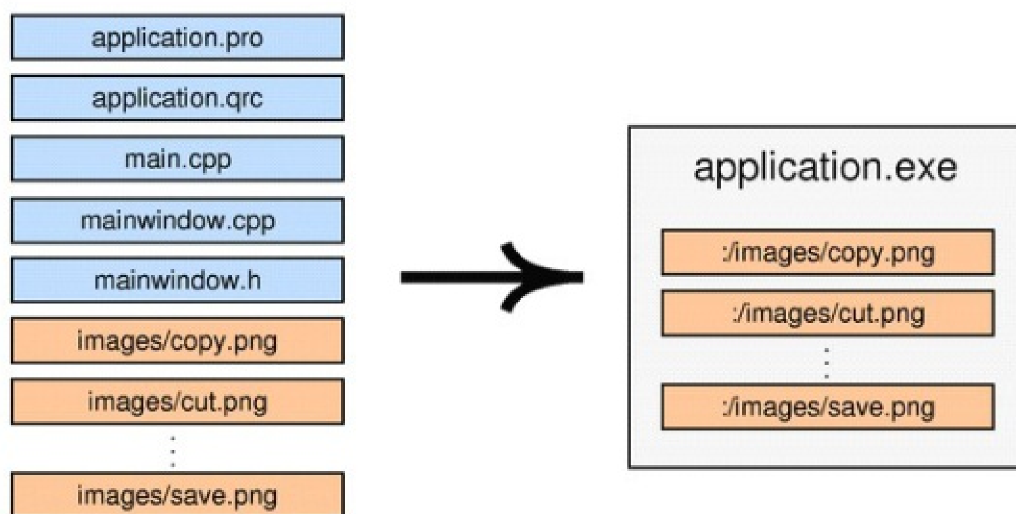


图 8-23 Qt 资源的编译

Qt 资源系统所具备并提供的功能远远不止我们所介绍的这些，它还包括对文件名别名（Alias）的支持和本地化（Locale）的支持。如果你对此感兴趣，请在 Qt Assistant 中 查阅“The Qt Resource System”一节中的内容。

目前，Qt4 资源系统总是将资源文件数据直接存储在应用程序的可执行文件中，这确实 导致该文件体积偏大，这也是目前 Qt4 被人们所指摘的几个主要问题之一。众所周知， Windows 以及 Mac OS X 系统都提供了对资源的原生支持。在 Qt 的后续版本中，有望对这一 情况进行适应性修改。

8.6.4 资源浏览器（Resource Browser）的使用

资源浏览器是 Qt Designer 的常用组件之一。在使用 Qt Designer 创建的每一个界面 布局都可以拥有独立的资源集文件（ .qrc 文件）。

资源浏览器默认情况下已经打开并位于 Qt Designer 的右下角，如图 8-24 所示。



图 8-24 资源浏览器

在资源浏览器内，可以打开一个已经存在的资源集文件或者是创建新的资源集文件。 可以通过点击快捷按钮来编辑资源。表 8-4 列举了快捷按钮的功用。

表 8-4 资源浏览器快捷按钮的功用

图标	功用
	编辑资源文件
	重新载入资源文件

载入资源集文件后，可以创建或者是删除其中包含的资源文件。表 8-5 列举了【编辑 资源】对话框中各个功能按钮的功用。

表 8-5 编辑资源对话框中功能按钮的功用

图标	功用
	增加前缀
	增加资源文件
	移除资源文件或者前缀
	新建资源文件
	打开资源文件
	移除资源文件

注意，添加到资源集文件内的资源文件必须与资源集文件在同一个文件夹内或者是位 于它的子菜单下面。

图 8-25 显示的是一个添加资源集文件的范例。



图 8-25 编辑资源集文件

注意，Qt 资源集文件中对前缀的要求不是必需的。

8.7 锚接窗口

锚接窗口又被称作是停靠窗口，是指用户可以在工具栏区域内或区域间随意移动的窗 口。用户可以对停靠窗口解锁，使该窗口浮在应用程序顶部，也可以使窗口最小化。 锚接窗 口是由 QDockWidget 类提供的。通过 QDockWidget 实例化并添加窗体，可以创建自定义 锚 接窗口。如果锚接窗口占据水平区域（例如，在主窗口的顶部），那么窗体将会横向排列；如果占据垂直区域（例如，在主窗口的左侧），那么窗体将会纵向排列。 锚接区域可嵌套， 以允许锚接窗口堆叠为多行或多列。

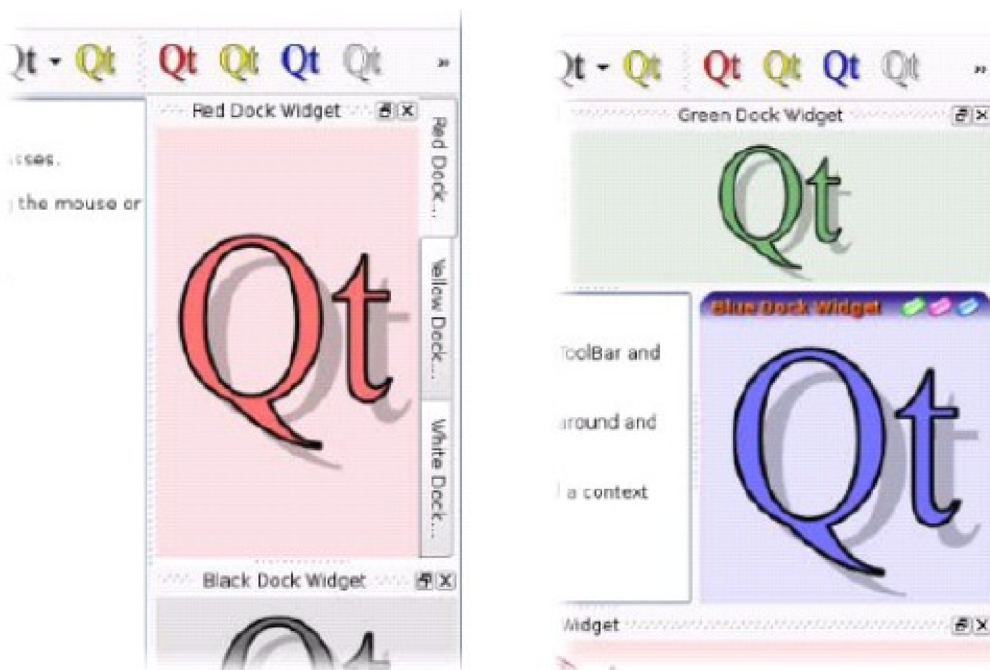


图 8-26 停靠区域包括三个停靠窗口，放在垂直标签页中； 其中一个停靠窗口带有自定义的标题栏和窗体控件。

停靠窗口可显示垂直的标题栏，窗口之前还可共享区域 – 当发生区域共享时，停靠 窗口将容纳在标签页中。 还可给停靠窗体设置风格独特的标题栏和窗体控件（参见上面的 图 8-26 ）

有些应用程序（包括 Qt Designer 和 Qt Linguist ）经常使用锚接窗口。 QMainWindow 为操作者提供保存并恢复锚接窗口和工具栏的位置的功能，这样，应用程序可 以轻松恢复用户首选工作环境。

8.7.1 创建锚接窗口的方法和流程

在主窗口程序中创建锚接窗口的一般流程如下：

1.创建锚接窗体

创建一个 QDockWidget 对象的锚接窗体。

2.设置此锚接窗体的属性

通常调用 setFeatures()及 setAllowedAreas()两种方法。

3.新建一个要插入锚接窗体的窗口部件

常用的一般为 QListWidget 和 QTextEdit。

4.把窗口部件插入锚接窗体

调用 QDockWidget 的 setWidget()方法。

5.在 MainWindow 中加入此停靠窗体

使用 addDockWidget()方法。 以下是一段示例代码，演示了上述方法和流程。

```
// 停靠窗口
QDockWidget *dock = new QDockWidget(tr("DockWindow"), this ); dock-&gt;setFeatures( QDock
dock-&gt;setAllowedAreas(Qt::LeftDockWidgetArea|Qt::RightDockWidgetArea); QTextEdit *te =
te-&gt;setText(tr("Dock Window!")); dock-&gt;setWidget( te );
addDockWidget( Qt::RightDockWidgetArea, dock );
```

8.7.2 设置锚接窗体状态的方法

主要是 setAllowedAreas()和 setFeatures()方法的使用。 其中，setAllowedAreas()方法设置停靠窗体可停靠的区域，原型如下：

```
void setAllowedAreas( Qt::DockWidgetAreas areas );
```

参数 areas 由 Qt::DockWidgetAreas 枚举变量指定了锚接窗体可停靠区域，包括表 8-6 列举的几种。

表 8-6 锚接窗体的可停靠区域

Qt::LeftDockWidgetArea	可在主窗口的左侧停靠
Qt::LeftDockWidgetArea	可在主窗口的右侧停靠
Qt::RightDockWidgetArea	可在主窗口的顶端停靠
Qt::BottomDockWidgetArea	可在主窗口的底部停靠
Qt::AllDockWidgetAreas	可在主窗口任意（以上 4 个）部位停靠
Qt::NoDockWidgetArea	只可停靠在插入处

以上各种情况可以叠加使用，采用或(|)的方式进行综合设定。 setFeatures()方法设置停靠窗体的特性，原型如下：

```
void setFeatures(DockWidgetFeatures features);
```

参数 features 由 QDockWidget::DockWidgetFeature 枚举变量指定锚接窗体的特性，

表 8-7 列举了所有的情况。

表 8-7 锚接窗体的特性

QDockWidget::DockWidgetClosable	停靠窗可关闭，右上角的关闭按钮
QDockWidget::DockWidgetMovable	停靠窗可移动
QDockWidget::DockWidgetFloatable	停靠窗可浮动
QDockWidget::DockWidgetFeatures	此参数表示拥有停靠窗的所有特性
QDockWidget::NoDockWidgetFeature	不可移动、不可关闭、不可浮动

此参数也可采用或(|)的方式对停靠窗进行特性的设定。

8.8 多文档

在每一个主窗口中只提供一个文档的应用程序被称为单文档界面（SDI）应用程序。基 于 SDI 的应用程序只提供了一个单一主窗口，并且在同一时间只能处理一个文档。如果想 让它在同一时间具有处理多个文档的能力，就需要同时启动多个应用程序实例。但是这对于 用户来讲是很不方便的。针对这种情况，我们可以使用基于多文档界面（MDI）的应用程 序，它也只有一个主窗口，但可以产生和管理多个文档窗口。基于多文档的应用程序也是很 常见的，比如 FireFox 浏览器就是典型的例子，它允许用户在同一时间内打开多个浏览器 窗口。

Qt 可以在它所支持的所有平台上创建 SDI 和 MDI 应用程序。 根据笔者的经验，在通常的工程项目中，使用 SDI 的应用程序占到了大多数的比例，但我们仍然需要对 MDI 有所了解。

8.8.1 创建多文档

在 Qt4 中创建 MDI 应用程序主要有以下方法：

1. 多实例实现主窗口的多文档

在一个应用程序中实例化多个主窗口，即当打开或新建文档的时候，文本编辑器应用 程序新建一个主窗口，这个主窗口单独加载和编辑文档。这种情况下，多个主窗口属于同一 个应用程序，当关闭所有的主窗口的时候，文本编辑器应用程序也就结束了运行。这种方法 称为多实例实现主窗口的多文档。

下面将修改应用程序，以使它可以处理多个文档。首先，需要对 File 菜单做一些简单 改动：

- 利用 File→New 创建一个空文档窗口，而不是再次使用已经存在的主窗口。
- 利用 File→Close 关闭当前主窗口。
- 利用 File→Exit 关闭所有窗口。

在 File 菜单的最初版本中，并没有 Close 选项，这只是因为当时它还和 Exit 一样具 有相同的功能。新的 File 菜单如图 8-27 所示。

新的 main()函数为：

```
int main(int argc,char *argv[])
{
    QApplication app(argc,argv); MainWindow *mainWin = new MainWindow; mainWin-&gt;show()
    return app.exec();
}
```

具有多窗口功能后，现在就需要使用菜单中的 new 来创建 MainWindow。考虑到节省内 存，可以再工作完成之后使用 delete 操作来删除主窗口。

这是新的 MainWindow::newFile()槽：

```
void MainWindow::newFile()
{
    MainWindow *mainWin = new MainWindow; mainWin-&gt;show();
}
```

我们只创建了一个新的 MainWindow 实例。这看起来有些奇怪，因为没有保留指向这个 新窗口的任何指针，但实际上这并不是什么问题，因为 Qt 会对所有的窗口进行跟踪。

以下是用于 Close 和 Exit 的动作：

```
void MainWindow::createActions()
{
    ...
    closeAct = new QAction(tr("Cl&ose"), this); closeAct-&gt;setShortcut(tr("Ctrl+F4"));
    closeAct-&gt;setStatusTip(tr("Close the active window"));
    connect(closeAct, SIGNAL(triggered()),mdiArea, SLOT(closeActiveSubWindow()));
    exitAct = new QAction(tr("E&xit"), this); exitAct-&gt;setShortcut(tr("Ctrl+Q"));
    exitAct-&gt;setStatusTip(tr("Exit the application"));
    connect(exitAct, SIGNAL(triggered()), qApp, SLOT(closeAllWindows()));
}
```

到此，我们就实现了从 SDI 到 MDI 的“转变”，组合好你的工程文件，编译、链接、 运行程序即可。

小贴士：在 Mac OS X 系统中，通常采用这种方法实现 MDI 应用程序。

2.使用 QWorkSpace

使用 QWorkSpace 作为主窗口的中心部件，在 QWorkspace 中打开多个子窗口，每一个 子窗口可以单独对文档进行加载和编辑。

QWorkspace 类继承自 QWidget 类，利用它可以很方便的实现多文档的应用。使用它的 方法有如下步骤：

第 1 步，在主窗口的头文件（如 mainwindow.h）中加入 QWorkspace 类的头文件，代码 如下：

```
#include &lt;QWorkspace&gt;
```

也可以采用类的前置声明，代码如下：

```
class QWorkspace;
```

第 2 步，在主窗口头文件中声明一个 QWorkspace 类对象，代码如下：

```
QWorkspace *workspace;
```

第 3 步，在构造函数中实例化该对象，并把该对象设置为主窗口的中心窗口部件，代 码如下：

```
MainWindow::MainWindow()
{
    workspace = new QWorkspace; setCentralWidget(workspace);
    ...
}
```

第 4 步，创建多个子窗口，并创建它们各自的中心窗口部件 。

```
    QMainWindow *window1 = new QMainWindow;
    window1-&gt;setWindowTitle(tr("window 1"));
    QTextEdit *textEdit1 = new QTextEdit;
    textEdit1-&gt;setText(tr("Window 1"));
    window1-&gt;setCentralWidget(textEdit1);

    QMainWindow *window2 = new QMainWindow;
    window2-&gt;setWindowTitle(tr("window 2"));
    QTextEdit * textEdit2 = new QTextEdit;
    textEdit2-&gt;setText(tr("Window 2"));
    window2-&gt;setCentralWidget(textEdit2);

    QMainWindow *window3 = new QMainWindow;
    window3-&gt;setWindowTitle(tr("window 3"));
    QTextEdit * textEdit3 = new QTextEdit;

    textEdit3-&gt;setText(tr("Window 3"));
    window3-&gt;setCentralWidget(textEdit3);
```

第 5 步，在 workSpace 对象中加入这几个子窗口，对它们进行管理，代码如下：

```
workSpace-&gt;addWindow(window1);
workSpace-&gt;addWindow(window2);
workSpace-&gt;addWindow(window3);
```

第 6 步，组织好你的工程文件，编译、链接、运行程序即可。 完整的示例程序在本章目录下的 workspace 文件夹下面。

小贴士：在 Qt4.5 版推出以后，不推荐采用上述方法来创建 MDI 应用程序。来自 Qt 的 官方说法是，QWorkspace 是被废弃的类，它的存在就是为了使采用以前版本的 Qt 开发的程 序能够正常运行。所以，如果你使用的是 Qt4.5 及以后的版本，我们强烈建议你使用 QMdiArea 来创建 MDI 应用程序。

3.使用 QMdiArea

这种方法的核心主要是掌握两个类的用法： QMdiArea 和 QMdiSubWindow，前者主要用 于创建程序主窗口的中心窗口部件，后者用于创建主窗口的各个子窗口。具体的做法是把 QMdiArea 类的实例作为主窗口的中心部件，把 QMdiSubWindow 类的实例作为子窗口，并由 QMdiArea 实现对多个子窗口的管理。

QMdiArea 类继承自 QAbstractScrollArea，它是 Qt 4.3 以后新增加的类。在创建 MDI 应用程序时，QMdiArea 类的实例通常被用作主窗口的中心窗口部件，但也可以被放置于一 个布局中。实际上，QMdiArea 是 MDI 应用程序的窗口管理器。它建立、绘制、管理在它之 上的子窗口，并可采用层叠或者平铺的方式排列它们。

QMdiSubWindow 继承自 QWidget，它的作用是为 QMdiArea 创建子窗口。它代表了在 QMdiArea 中创建的顶层窗口。它主要包含一个标题栏、一个内部窗口（ Internal Widget）、一个窗口框架和一个大小控制手柄。 QMdiSubWindow 有自己的布局（Layout）， 在其中包含窗口标题栏以及内部窗口的中心窗口区域。一个典型的 QMdiSubWindow 实例如 图 8-27 所示。

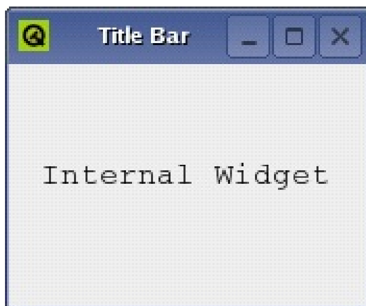


图 8-27 QMdiSubWindow 子窗口

听了上面的介绍，大家是不是有点“晕”呢？别着急，请看图 8-28，它示意了采用这 种方法创建 MDI 应用程序的窗口布局以及创建的方法。图中的主窗口是子类化 QMainWindow 类创建的；主窗口的中心窗口部件使用 QMdiArea 的实例创建；子窗口是子类化 QWidget 实 现的，而子窗口的内部窗口部件是使用 QMdiSubWindow 的实例创建的。

它们之间的关系总结如下：QMdiArea 是所有子窗口的容器和管理器，QMdiArea 中的子 窗口都是 QMdiSubWindow 类的实例。我们通过 addSubWindow()方法把它们加入到 MDI 应用 程序中。使用时，通常先建立一个 QWidget 或其子类的实例，然后把它作为参数调用 addSubWindow()函数，addSubWindow()函数将把它作为子窗口的内部窗口，并填充中心窗口 区域。由于 QMdiSubWindow 是 QWidget 的子类，所以你可以像使用以前我们介绍过的常见 顶层窗口那样使用它，如可以调用基类 QWidget 的 show(), hide(), showMaximized(), 以及 setWindowTitle()等方法对窗口实例进行设置。

看着这张图再对照笔者上面的讲解，应该就很清楚了吧。



图 8-28 使用 QMdiArea 类创建 MDI 时的窗口框架

小贴士：为 QMdiSubWindow 创建内部窗口有两种方法，一种是调用 addSubWindow(widget)，其中 widget 参数将作为内部窗口部件；另一种是先创建一个继承自 QWidget 的窗口实例，然后调用 setWidget (QWidget * widget)方法，把 widget 作为子窗口的内部窗口部件即可，这个内部窗口部件将被显示在子窗口的中心区域。注意，QMdiArea 会对其内部的子窗口进行管理，你不必使用代码显式的管理它们。

QMdiSubWindow 还有许多专门对应 MDI 应用类型而设置的方法和属性，大家可以在 Qt Assistant 中获得详尽的介绍。

好了，我们详细解说一下如何采用 QMdiArea 和 QMdiSubWindow 类来创建 MDI 应用程序，这个例子在本章目录 mdi 文件夹下面。

第 1 步，包含用到的类

在主窗口的头文件（如 mainwindow.h）中包含程序中使用到的类,有两种方法，一是可 以加入头文件；二是当情况比较简单时，也可以采用类的前置声明。

加入头文件：

```
#include &lt;QMdiArea&gt;
#include &lt;QMdiSubWindow&gt;
...
```

或者采用类前置声明：

```
class QMdiArea;
class QMdiSubWindow;
...
```

第 2 步，声明一个 QMdiArea 类对象

在主窗口的头文件中声明一个 QMdiArea 类对象，在后面还需要声明一个你的子窗口类 的对象，代码如下：

```
QMdiArea *mdiArea;
...
MdiChild *child;//声明子窗口类的对象
...
```

第 3 步，设置中心窗口部件

在主窗口类的实现文件中（如 mainwindow.cpp，通常在其构造函数中）实例化该对 象，并把它设置为主窗口的中心窗口部件，代码如下：

```
MainWindow::MainWindow()
{
    mdiArea = new QMdiArea; setCentralWidget( mdiArea );
    ...
}
```

第 4 步，创建子窗口

新建一个子窗口类，它可派生自 QWidget 或其子类，比如 QTextEdit。这个类的实例将 作为子窗口的内部窗口部件。这个子窗口类的创建与我们前面讲到的子类化对话框和子类化 QWidget 的方法相同，只是它没有菜单栏、工具栏和状态栏。

另外记得在主窗口的头文件中加入该子窗口类的声明。

第 5 步，实例化子窗口类，并使用 QMdiArea 对它进行管理，代码如下：

```
child = new MdiChild;
QMdiSubWindow *subWindow = mdiArea-&gt;addSubWindow(child); subWindow-&gt;show();
...
```

小贴士：QMdiArea::addSubWindow()函数创建一个新的 QMdiSubWindow，把作为参数传递的 该窗口部件放进子窗口中，并且返回该子窗口。最后一行代码调用 show()方法，使该子窗口可见。

第 6 步，创建并显示子窗口

这通常是在用户点击 File->NewFile 时完成的，代码如下：

```
void MainWindow::newFile()
{
    child-&gt;newFile(); child-&gt;show();
}
```

整个程序的实现过程可以用图描述。

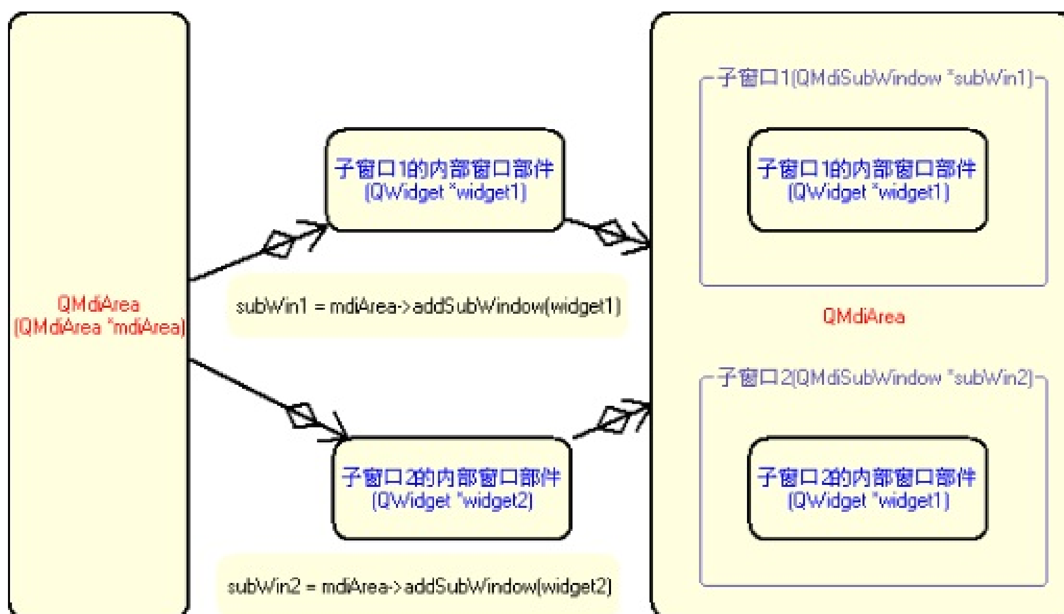


图 8-29 程序的实现过程

小贴士：在 MDI 应用程序中，主窗口类并不需要对文档进行具体处理，这些工作是在子窗 口类中完成的，相当于在 SDI 应用程序中实现的文档处理功能。

第 7 步，创建 main.cpp 文件 这步没有什么好说的，代码如下：

```
#include &lt;QApplication&gt;
#include "mainwindow.h"
int main(int argc, char *argv[])
{
    Q_INIT_RESOURCE(mdi);//使用 Qt 资源系统
    QApplication app(argc, argv); MainWindow mainWin; mainWin.show();
    return app.exec();
}
```

做完这些后，组织好你的工程内的文件（头文件、实现文件、资源文件、资源集文件 等），编译、链接、运行程序即可。一个典型的 MDI 应用程序界面如图 8-30 所示。

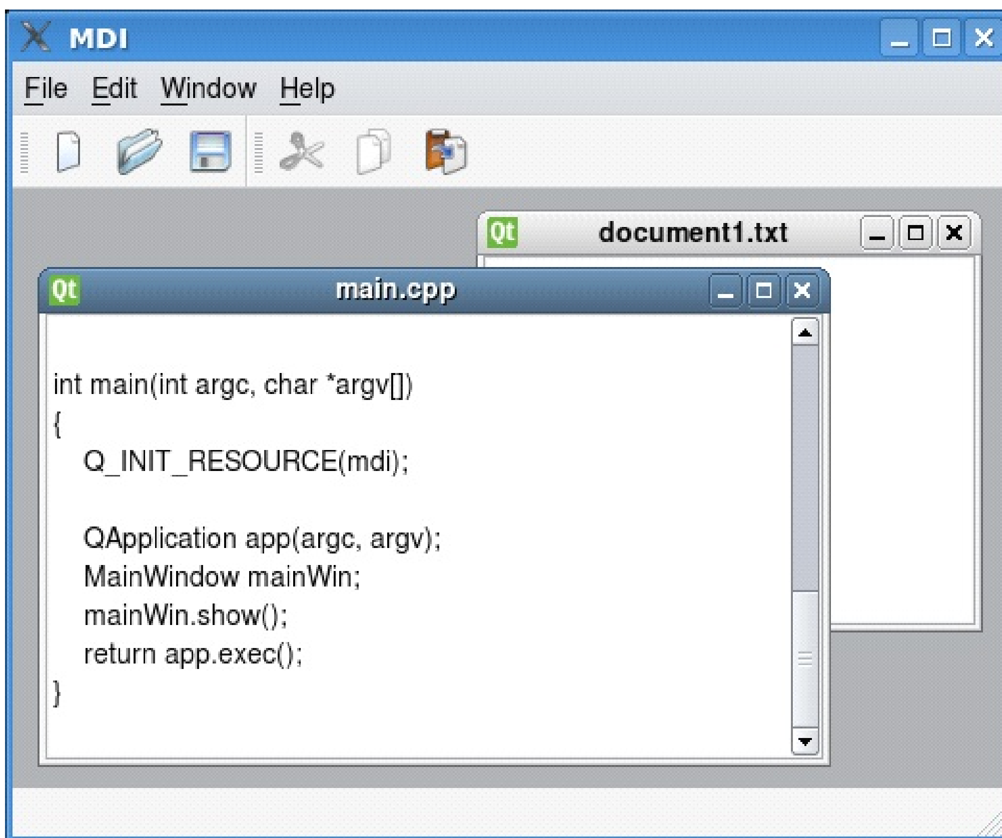


图 8-30 使用 QMdiArea 建立的 MDI 应用程序界面

到这里，关于如何使用 Qt4 创建 MDI 应用程序就讲解完了。采用 QMdiArea 的方法是笔 者重点讲解的，一是因为它是 Qt4.5 版以后官方所推荐采用的方法，用于替代 QWorkspace；二是因为它也是实现 MDI 的方法中使用起来最为复杂和令人困惑的一个。即 使是让 Qt“老鸟”来解释清楚什么是“主窗口的中心窗口”、“子窗口的内部窗口”、 “子窗口的中心区域”等等这些名词以及用法，也不是一件容易的事。

相对而言，使用 QWorkspace 创建 MDI 是比较容易的，但 Qt 以后将不再对这个类继续更新和支持，而“多实例实现多文档”的方法在 Mac OS X 上应用很广泛，它实质上就是使 用了多个顶层窗口，也是比较容易掌握的。

熟练使用 Qt4 建立 MDI 应用程序所涉及的内容远远不止本书所介绍的这些，比如子窗 口如何响应键盘与鼠标事件，如何同步所有主窗口的 “最近打开文件列表”等等问题都需要 费些力气解决。对于初学者而言，这是一个比较复杂的话题。大家在阅读到此处时，建议仍 然采用“知道、会用即可”的原则，不必深究它们背后的机理。随着学习进程的逐步深入， 一些开始接触时觉得困难的内容，就会在你心中逐渐明晰了。

8.9 问题与解答

问：我用的是 Qt Designer，会自动加上 statusbar，如何把它去掉呢？

答：如图 8-31 所示，可以在对象查看器（Object Inspector）中点击鼠标右键，然后 把它删掉。

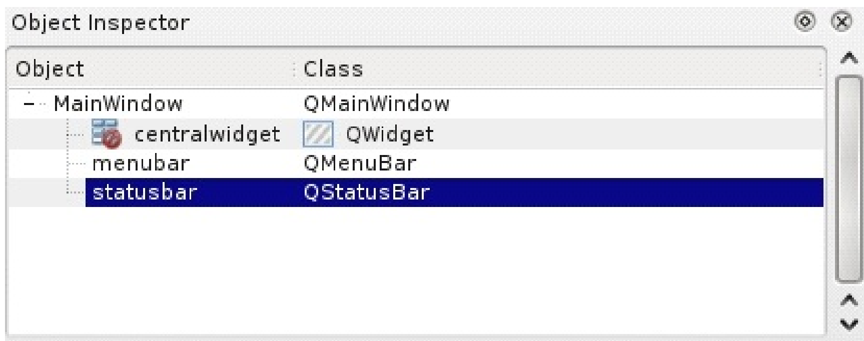


图 8-31 在对象查看器中删除状态栏

对象查看器（Object Inspector）是很有用的一个部件工具，可以索引窗口部件。

问：在 Qt4 中如何创建有最大最小化的 MainWindow？

怎么新创建的 MainWindow 就只有一个关闭按钮,没有最大最小化呢,而且拖边框放大缩 小也不可以,只能在右下那个角拖.

答：用 setWindowFlags()方法设置。在 Qt Demos 里面有一个例子是演示 setWindowFlags 里各种选项的效果的。

你的问题可以使用下面的语句：

```
setWindowFlags(windowFlags() & ~Qt::WindowMinMaxButtonsHint);
```

问：怎样去掉 MainWindow 最大化和最小化按钮？

答：这个问题与前一个类似，用 setWindowFlags()方法设置。

```
setWindowFlags(Qt::CustomizeWindowHint ); //这样应该能够达到你的需要
main_window* window
    = new main_window(this,0,FALSE,WStyle_Customize | WStyle_NormalBorder | WStyle_Title
```

也可以这样写：

或者在构造函数中加上这两句：

```
setWindowFlags(Qt::Dialog); //窗体没有最大化最小化按钮
setFixedSize(250, 100); //设置窗体的大小
```

问：MainWindow 窗口如何固定？

项目要求将 MainWindow 窗口的标题栏屏蔽。就是只响应它的关闭按钮，对于最大化和 最小化都不响应，而且也不能被鼠标拖动， 请问有没有什么好的办法。

答：以下代码可以达到效果，可以看做是前面几个问题的综合示例：

```
#include &lt;QtGui&gt;
#include &lt;QtCore&gt;
int main(int argc, char **argv)
{
    QApplication app(argc, argv); QMainWindow window;
    window.setWindowFlags(window.windowFlags()
        & ~Qt::WindowMaximizeButtonHint
        & ~Qt::WindowMinimizeButtonHint
    );
    int titleBarHeight =
        QApplication::style()-&gt;pixelMetric(QStyle::PM_TitleBarHeight);
    QRect rect = QApplication::desktop()-&gt;availableGeometry();
    rect.setTop(rect.top() + titleBarHeight);
    window.setGeometry(rect);
    window.setFixedSize(window.size());
    window.showMaximized();
    return app.exec();
}
```

style()、setGeometry()、setFixedSize()方法在涉及到界面布局时经常会用到，用 法比较简单，可以查阅 Qt Assistant。

问：在 Qt Designer 中设计 Main Window 程序的问题

我在 Main Window 里放置了一个 TextEdit，被自动设为 centralWidget。可是在程序 执行的时候，该 TextEdit 不能充满整个程序界面，请问如何解决？

答：你对中心部件（Central Widget）的认识是存在误区的。那个 TextEdit 是不会被 自动设置为中心部件的。如果使用 Qt Designer 的话，你要添加一个布局管理器， 然后设 置主窗口的中心部件，其实和手写代码是一样的。

8.10 总结与提高

本章主要介绍了以下内容：

- 应用程序主窗口框架的组成
- 常用的创建主窗口的方法和适用场合
- 完全使用代码创建主窗口的方法和步骤
- 使用 Qt Designer 和代码相结合创建主窗口的方法和步骤
- 中心窗口部件专题讲解
- Qt4 资源系统专题讲解
- 创建多文档应用程序的方法和步骤 这些都是日后在工程开发实践中经常会用到的基本知识和技能，必须熟练掌握。对于多文档的应用，建议大家只要能够熟练运用本书中介绍的 3 种方法中的一种就可以了。

关于主窗口的应用还有很多更为深入的内容，但它们都是建立在本章所介绍内容的基 础上的，大家把基础打好，再一步一步的向上 “攀登”，就会容易多了。

最后，提出一个问题供大家思考，即如何实现在 Qt 的工具栏中嵌入其他的窗口部件， 比如列表框(QComboBox)、编辑框(QLineEdit)。这种情况是很常见的，在 MS Word、IE 中 都有很多类似的应用。提示一下，我们前面提到过 Qt 编程的一般顺序，就是先声明所需的 窗口部件，然后实例化它，定义它的实例的属性和方法，最后就是在需要的地方使用它。这 个问题，我们也是这样处理，最后使用 QToolBar 的 AddWidget()方法把窗口部件的实例加 入到工具栏中。请大家自行完成这个 练习。

第 9 章 Qt 样式表与应用程序观感

本章重点

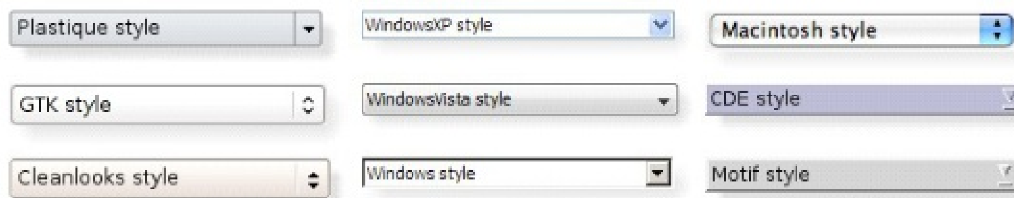
- 掌握设置应用程序观感的方法
- 了解 Qt 窗口部件的样式
- 了解 Qt 样式表的作用
- 掌握 Qt 样式表的基本语法
- 掌握样式表与 Qt Designr 的集成使用
- 了解子类化 QStyle 类的方法
- 掌握不使用样式表改变应用程序观感的方法

Qt 样式表是专为设置应用程序的观感（ look and feel）而生的，它是从 Qt 4.2 开始引 入的描述窗口部件观感的强大机制 ，允许你按照自己的需求定制应用程序的观感 。并且从 Qt 4.5 开始，Qt 样式表全面支持 Mac OS X 平台。

Qt 样式表的设计灵感主要来自于 HTML 的层叠样式表（Cascading Style Sheets,CSS）， 但它同时适用于窗口部件。在介绍 Qt 样式表之前，我们先来看看什么是应用程序的外观（或 者叫做观感）。

9.1 应用程序的观感

相信大家已经发现，Qt 应用程序在每一个所支持的平台上都可以看起来像原生的本地程 序一样。Qt 是通过模拟各个平台的观感来实现这一点的， 而不是使用特定的平台或者工具包 的窗口部件集。图 9-1 显示了在不同平台下的 Qt 组件中的 ComboBox 的观感。



9.1.1 应用程序的风格

图 9-1 窗口部件在不同平台上的观感

每个应用程序都有自己的样式（Style）。运行于 KDE 下的 Qt/X11 应用程序的默认样式 是 Plastique，而运行于 GNOME 下的应用程序的默认样式是 Cleanlooks。这些样式使用了渐变和抗锯齿效果，以用来提供一种时尚的观感。运行 Qt 应用程序的用户可以通过使用命令行参数-style 覆盖原有的默认样式。例如，在 X11 下，要想使用 Motif 样式来运行名为 app 的程序，只需要简单的输入以下命令即可：

```
./app -style motif
```

与其他样式不同，Windows XP、Windows Vista 和 Mac 所特有的样式只能在它们的本地平台上有效，因为它们需要依赖相应平台的主题引擎。

还有另外的一种样式 QtDotNet，它来自于 Qt Solutions 模块。你也可以创建出自己所 特别喜好的与众不同的样式，这部分内容已经超出了本书讨论的范围，如有需要请参阅相关 文档。

9.1.2 如何设置样式

要设置整个应用程序的样式，可以调用 QApplication::setStyle()方法，或者由用户在 应用程序运行时输入-style 命令行参数来指定：

```
./myapplication -style motif
```

如果没有显式的指定，那么 Qt 将根据用户的平台和桌面环境自行选择最为合适的样式 。

如果要单独设置某个窗口部件的样式，可以对其调用 QWidget::setStyle()方法。

9.2 QStyle 类的使用

说到样式表，就不可避免的要谈到 Qt 的样式（Qt Style）。

在 Qt 样式表出现之前，我们主要是通过子类化 QStyle 类或者预定义一个样式，例如 QWindowStyle，来定制应用程序的观感，而 Qt 本身也是使用这种方法来为它所支持的不同平 台提供特定的观感的。

QStyle 类的应用比较广泛。它封装了 GUI 应用程序所使用的观感样式，是一个抽象基类。Qt 包含了一系列的 QStyle 的子类，它们模拟了在各种不同平台上的 GUI 界面的观感，这些观感可以通过设置 QWindowStyle、QMacStyle、QMotifStyle 等样式来实现,而且在我们调用 QtGui 这个模块时，如果事先没有显式的指定，那么 Qt 会自动判断环境并选择最为适合的样式。此外，样式也可作为插件来被应用程序使用。

下面我们使用 Qt Creator 结合代码，创建一个实例，大家从中可以学习到使用 QStyle 类配置应用程序的观感的方法。

9.2.1 实例-使用 QStyle

本实例的代码见 styleCreator 实例。

第 1 步，创建项目。

启动 Qt Creator，在弹出的【New...】对话框中选择 Empty Qt4 Project 类型，如图所 示，点击【OK】按钮进入到下一步的设置界面。

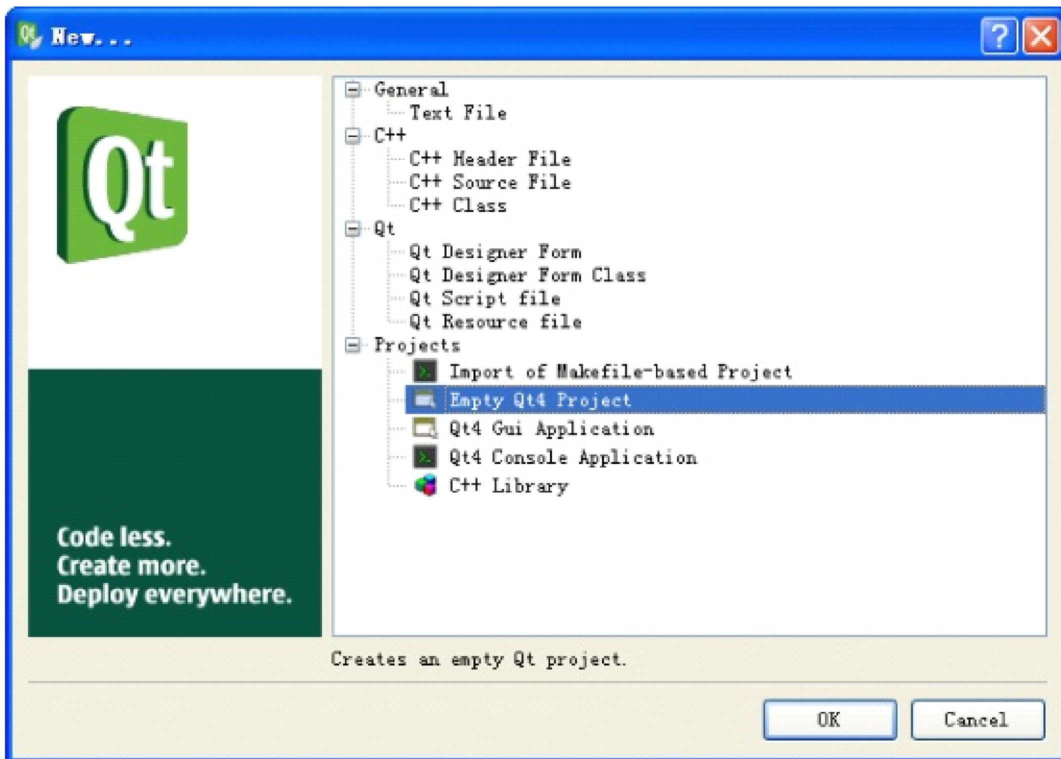


图 9-2 选择项目类型

然后，在弹出的【Empty Qt4 Project】对话框里为项目设置名称和保存位置，这里项 目名称为 styleCreator。注意，名称和保存的位置中都不要含有空格、特殊字符和中文字符。 图 9-3 和图 9-4 显示了这个设置的过程。

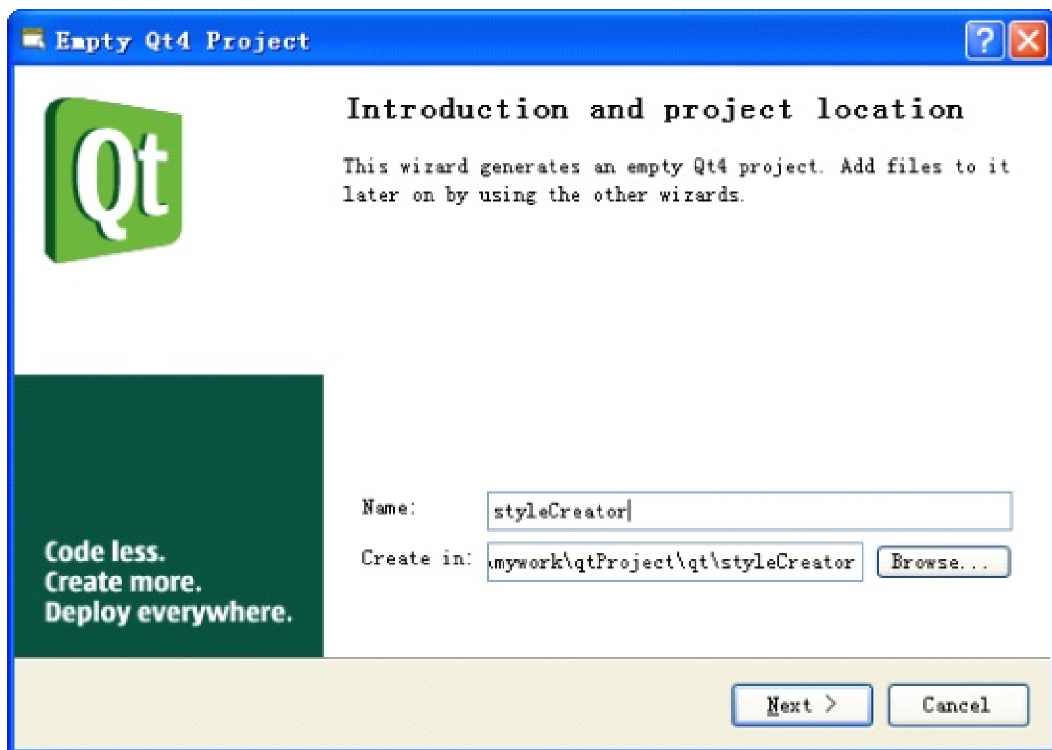


图 9-3 设置项目名称和要保存的位置



图 9-4 设置项目保存的位置

这之后，一路选择默认设置即可完成项目的创建。如果读者朋友对使用 Qt Creator 的 使用有不熟悉的地方，可以参见第 6 章和第 12 章。

第 2 步，设计界面。

我们在 Qt Designer 中设计出来的界面布局如图 9-5 所示，具体的布局结构可以从对象 查看器中得到，不再赘述。

界面设置完成后，将其保存在第 1 步创建的项目目录下面，名为 styleCreator.ui。

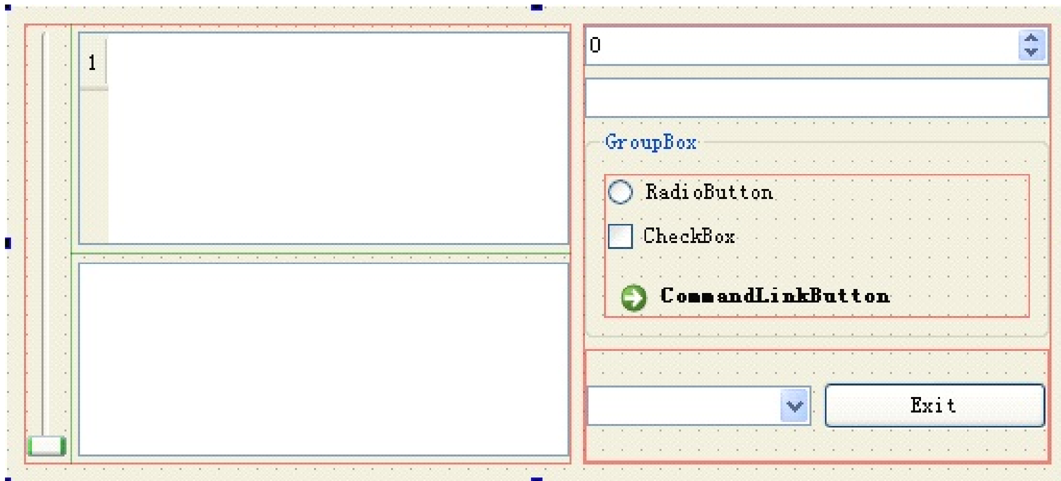


图 9-5 界面布局

第 3 步，书写并向项目中添加文件。

本项目中共有 styleCreator.pro、main.cpp、myWidget.h、myWidget.cpp、和 styleCreator.h 这几个原生源文件。首先要向项目中加入这几个文件，一般有两种方法 。一种是直接在 Qt Creator 中定义、书写并加入；一种是在 Qt Creator 之外书写好，然后再加入进来。我们以 main.cpp 为例，简要介绍这两种方法。

第一种方法介绍如下，如图 9-6 所示，首先依次点击【File】→【New】，在弹出的【New...】 对话框中选择要新建的类型，这里是 C++ Source File。然后，点击【OK】按钮，进入到下 一步设置。

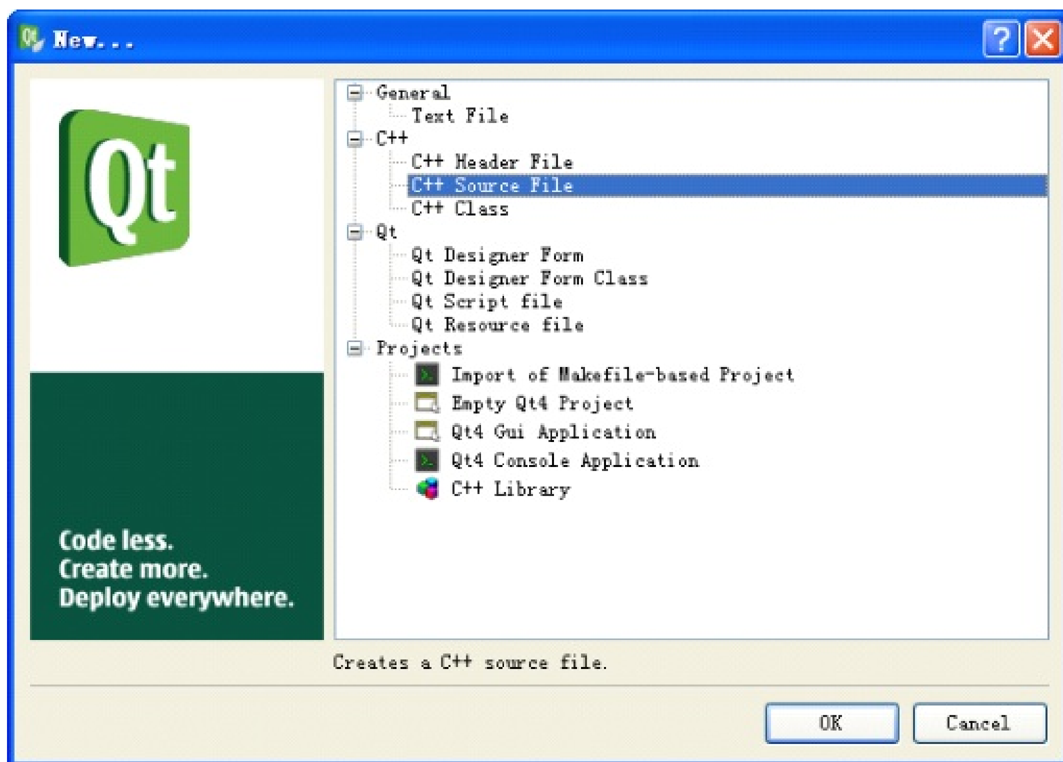


图 9-6 新增 C++源文件

在接下来弹出的【Chose the location】对话框中设置新建文件的名称和要保存的位置， 这里要选中第 1 步创建好的项目的位置。如图 9-7 所示。点击【Next】按钮进入下一步。

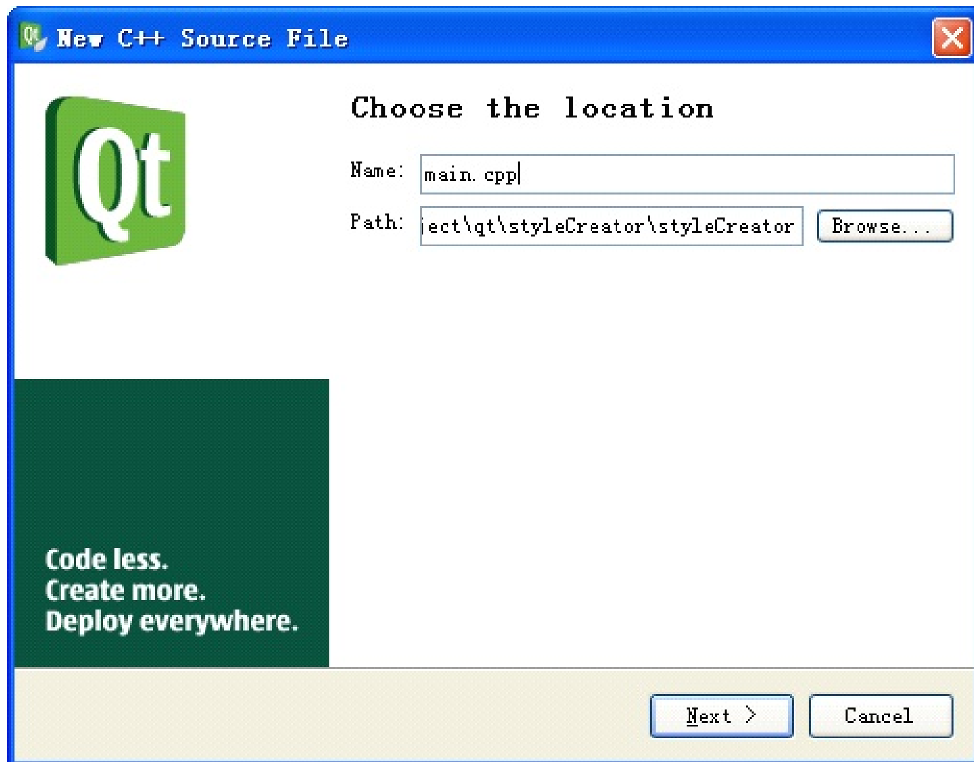


图 9-7 选择文件的位置

接下来，在弹出的【Project management】对话框中选中那个 Add to Project 复选框， 并选择第 1 步创建好的项目 Project 名称。如图 9-8 所示，点击【Finish】按钮完成新文件 的添加。

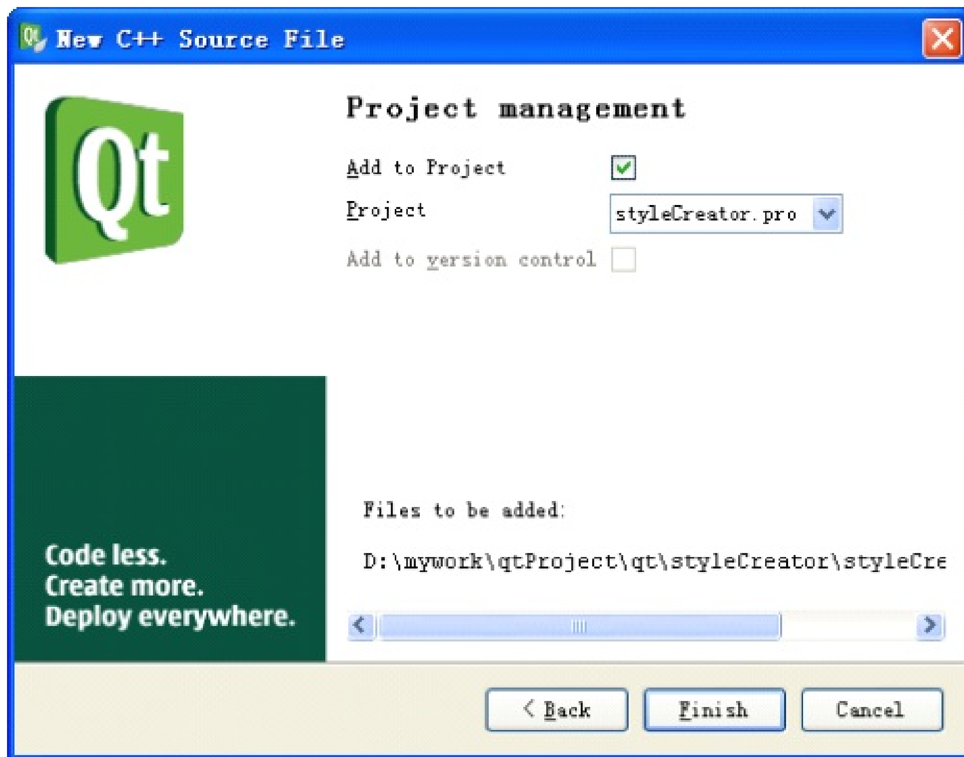


图 9-8 选择项目名称

第 2 种方法介绍如下，如图 9-9 所示，在 Qt Creator 中的项目栏上点击鼠标右键，选 择 Add Existing Files...菜单项，将你在项目外面书写好的文件加入到项目中。对于前面 使用 Qt Designer 做好的 styleCreator.ui 文件，我们使用这种方法把它加入到项目中。

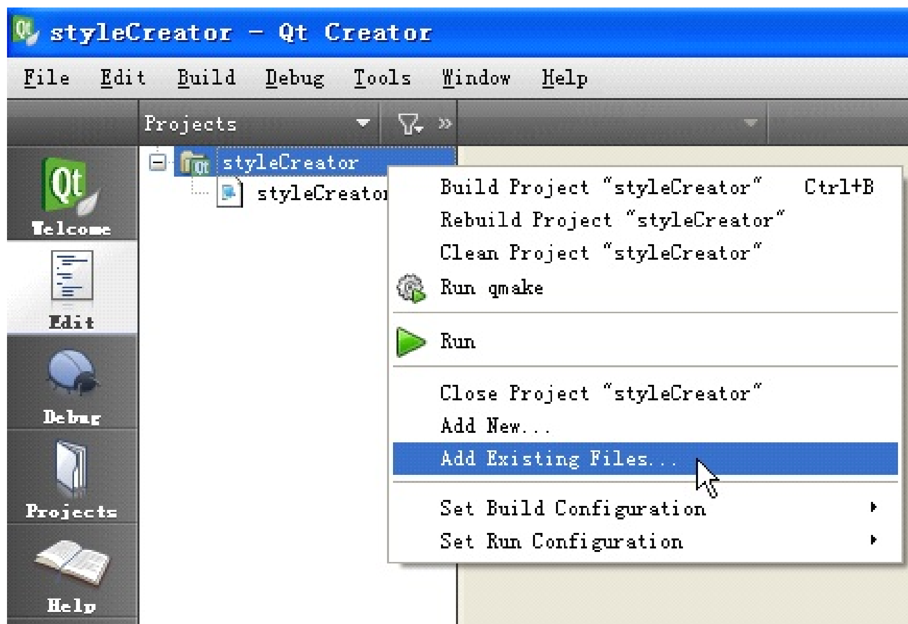


图 9-9 使用右键菜单添加文件

第 3 步，书写代码。 我们选择使用多继承的方法，从原生界面文件实体类中派生出一个自定义类 myWidget，首先看一下头文件。

```
#ifndef MYWIDGET_H
#define MYWIDGET_H
#include &lt;QDialog&gt;
#include "ui_styleCreator.h"
class myWidget:public QDialog,public Ui::Dialog
{
    Q_OBJECT
public:
    myWidget( QDialog *parent = 0 );
public slots:
    void slotChangeStyle(QString);
};
#endif // MYWIDGET_H
```

首先要包含 QDialog 类的声明，接下来再加入对 ui_styleCreator.h 的包含。

注意声明 myWidget 类的方法，是多继承自 QDialog 类和定义在 ui_styleCreator.h 中定义的 Ui::Dialog 这个实体类。

由于程序中要用到 Qt 的核心机制-信号/槽，所以必须加上 Q_OBJECT 这个宏。然后声明 构造函数和一个公有槽。

再来看一下类的定义文件。

```
#include "myWidget.h"
#include &lt;QtGui&gt;
myWidget::myWidget(QDialog *parent)
:QDialog(parent)
{
    setupUi(this);
    setWindowTitle(tr("Change Window Look And Feel"));
    comboBox-&gt;addItems(QStyleFactory::keys());
    this-&gt;spinBox-&gt;setRange(0,9);
    this-&gt;lineEdit-&gt;setText(tr("Hello Qt Style,We Can Change It!"));
    this-&gt;lineEdit-&gt;setReadOnly(true);
    this-&gt;textEdit-&gt;setText("Hello Qt!\n Qt by Nokia is the standard framework for
    this-&gt;tableWidget-&gt;setRowCount(3);
    tableWidget-&gt;setColumnCount(3);
    connect(comboBox,SIGNAL(activated(QString)),this,SLOT(slotChangeStyle(QString)));
    slotChangeStyle(QStyleFactory::keys()[5]);
}
void myWidget::slotChangeStyle(QString style)
{
    QApplication::setStyle(QStyleFactory::create(style));
    QApplication::setPalette(QApplication::style()-&gt;standardPalette());
}
```

下面我们讲解一下这些代码的作用。

第 1、2 两行引入头文件。

第 5 行初始化界面布局，该句一般放在构造函数的开头。

第 6 行设置程序标题，注意在字符串前面要用到 tr()函数，使程序满足国际化的要求。 第 7 行向组合框中加入具体的项值。

QStyleFactory 类用来创建 QStyle 对象。使用 QStyleFactory 的静态方法 QStyleFactory::keys()可以获得与具体平台无关的内置的样式，以及与平台有关的部分样式。与平台无关的样式主要包括"windows", "motif", "cde", "plastique" 和"cleanlooks"等风格，与平台有关的主要包括"windowsxp", "windowsvista" 和"macintosh"等几种样式。 此外，书写样式时，注意它们的名称是大小写敏感的。

第 8 行为滑块部件设定滑动的范围。

第 9 行设置编辑框要显示的文本内容。

第 10 行设置编辑框内容为只读。

第 11 行设置文本框显示的文本内容。

第 12 行设置表格部件的行数为 3 行。

第 13 行设置表格部件的列数为 3 列。

第 14 行连接组合框的触发信号和窗体的 slotChangeStyle 槽。 第 15 行像调用普通函数一样，调用 slotChangeStyle 槽函数。 第 17-21 行是 slotChangeStyle 槽的定义。

其中第 19 行通过调用 QApplication::setStyle()方法来设置应用程序的样式，该方法 是 QApplication 类的静态函数，其原型如下：

```
void QApplication::setStyle ( QStyle * style ) [static]
```

该方法的形参是 QStyle 类的对象，在本程序中，它的实参是通过

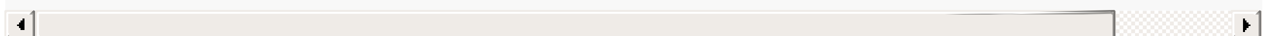
QStyleFactory::create()方法来获得的。该方法原型如下：

```
QStyle * QStyleFactory::create ( const QString & key ) [static]
```

其形参是 QString 的对象，返回值是 QStyle 的对象，该方法是 QStyleFactory 类的静态方法。

第 20 行调用 QApplication::setPalette()方法设置应用程序的颜色组合。该方法是 QApplication 类的静态方法，其原型如下：

```
void QApplication::setPalette ( const QPalette & palette, const char * className = 0 ) [s
```



它的作用是将应用程序原有的颜色组合用 palette 来替换。如果 className 被指定不为空，那么该方法将仅对该参数对应的窗口部件起作用；如果 className 取默认值 0，则 该方法将影响整个儿应用程序中的窗体部件的颜色组合，并且将之前在应用程序中设定的颜色组合替换掉。

最后像下面这样书写 main.cpp 文件内容，主要是将自定义的窗口部件显示出来。

```
#include &lt;QApplication&gt;
#include &lt;QWidget&gt;
#include"ui_styleCreator.h"
#include"myWidget.h"
int main(int argc, char * argv[])
{
    QApplication app(argc,argv);
    myWidget widget;
    widget.show();
    return app.exec();
}
```

通过这个实例，我们可以看到使用 QStyle 类来设置应用程序或窗口部件的样式是比较简便的。

在 Qt 应用程序中，QStyle 类的使用非常广泛。本章只是向大家介绍了经常会用到的基 本功能，其它的如如何创建一个自定义的样式、如何使用这个自定义的样式、怎样使自定义 的窗口部件感知这个样式等内容都是比较深入的话题，读者朋友可以在掌握本 节内容的基础 上，有选择的学习。

9.3 样式表概述

样式表的出现并不是要取代子类化 QStyle 类的方法。事实上，样式表在 Qt 的风格之上 起作用（如果使用了样式表，QWidget::style()返回的 QStyle 为”style sheet”），提供 了比 QPalette（用来设置窗口部件的颜色组合）更为灵活、强大的机制。

9.3.1 基本语法

样式表的语法和 HTML CSS 基本是一致的。Qt 的样式表对大小写不敏感，但对类名、对 象名和属性名大小写敏感。如下示例设置了所有 QTextEdit 对象背景是黄色的，所有 QPushBox 对象文本为绿色：

```
QTextEdit {background:yellow}
QPushBox {color:green}
```

1.样式规则

样式表包含一系列的规则，一个样式规则由选择符和定义组成。选择符（ selector）确 定有哪些窗口部件受规则影响，定义说明了在窗口部件上应用哪些属性。

例如：

```
QRadioButton {color:red}
```

在这条规则里，QRadioButton 是选择符，{color:red}是定义。这条规则说明了 QRadioButton 和它的子类应该使用红色作为前景色。

几个选择符可以使用一个定义，使用逗号分隔选择符。如：

```
QPushButton,QCheckEdit,QComboBox {color:red}
```

定义由一个或多个属性和值对组成，中间用分号隔开，如：

```
QPushButton { color:red;background-color:white }
```

2.选择符类型

Qt 支持所有 CSS2 中所有的选择符，表 9-1 给出了常用的选择符。

表 9-1 常用的选择符

选择符	示例	可以匹配的窗口部件
通配	*	所有窗口部件
类型	QLabel	给定类的实例，包括其子类
属性	QComboBox[editable="true"]	所有可以编辑的给定类的实例
类	.QCheckBox	给定类的实例，而不包括其子类
标识	QRadioButton#red	对象名为 red 的给定类的对象
子孙对象	QWidget QToolButton	所有是 QWidget 的子孙对象的 QToolButton 对象
子对象	QWidget>QGroupBox	所有是 QWidget 的直接子对象的 QGroupBox 对象

3.子控件

对于复杂控件，可以访问它的子控件。如 QCheckBox 上的下拉按钮，QSpinBox 上的向上和向下箭头。如：

```
QComboBox::drop-down { image:url (myarrow.png) }
```

上面的代码使用了自定义的下拉按钮图像。 ::是 CSS3 中的伪元素。

4.伪状态

选择符可以包含伪状态来表示窗口部件的状态。伪状态在选择符之后，以冒号分隔， 下面定义了当鼠标在 QPushButton 上悬停时的规则：

```
QPushButton : hover {color : white }
```

5.冲突解决

当不同的规则应用到相同的属性时，样式表就产生了冲突。在这种情况下，特定的规则 比通用的规则优先；伪状态比没有伪状态的优先；如果级别相同，则最后一个规则优先。冲 突解决按照 CSS2 规范进行。

6.层叠

样式表可以在 Qapplication 这个级别设置，也可以在父窗口部件，子窗口部件级别设 置。实际应用样式时，则合并者几个级别的样式。当有冲突时，窗口部件自身的样式优先使 用，接下来是父窗口部件，祖先窗口部件，依次类推。

7.盒子模型

窗口部件盒子窗口部件支持背景（background）、边框（border）、边距（margin）、填衬（padding），图 9-10 显示了样式表的盒子模型。

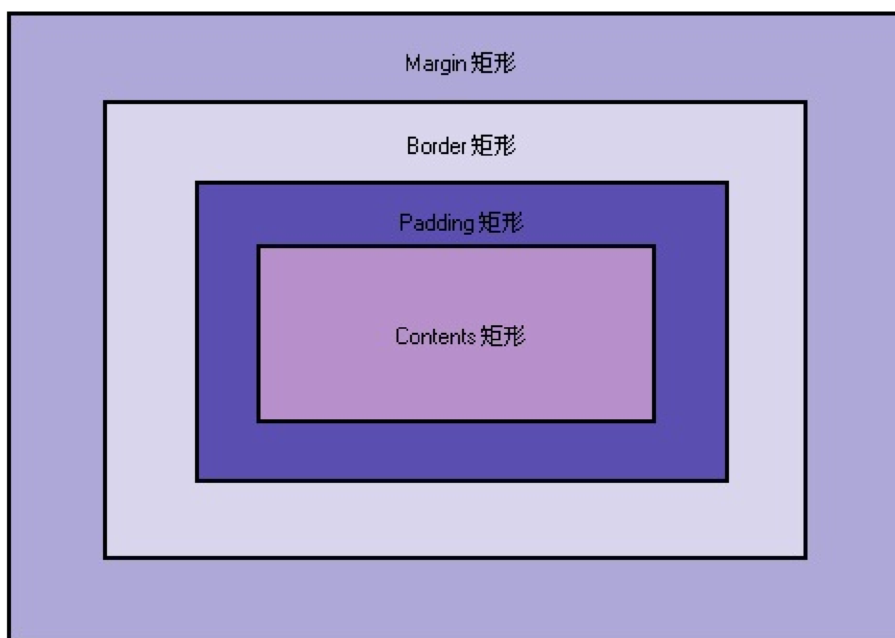


图 9-10 Qt 样式表的盒子模型

这个模型可以指定 4 个影响布局的矩形，从而绘制一个自定义的窗口部件：

(1) contents 矩形位于最里面。它是绘制窗口部件内容（如文字或图片）的地方。

(2) padding 矩形包围 contents 矩形。它负责由 padding 属性指定填充操作。

(3) border 矩形包围 padding 矩形。它为边界预留空间。

(4) margin 矩形在最外边，它包围 border 矩形，负责任何指定的边缘空白区域。 对于没有 padding、border 和 margin 的普通窗口部件，这 4 个矩形重合在一起。

9.4 使用样式表

使用样式表通常有两种做法，一种是设置全局的样式表，通过调用 QApplication::setStyleSheet()方法来实现；一种是在指定的窗口部件上使用样式表， 这可以通过调用 QWidget::setStyleSheet()方法来实现。如果你为同一个部件或应用程序指定了 多种不同的样式表（比如有的设置字体颜色，有的设置界面背景），那么 Qt 将会自动将它们 的效果组合起来，这也被称作是样式表的层叠（ cascading）。

注意，Qt 样式表目前尚不支持用户自定义的 QStyle 样式，在后续的版本中有望获得支 持。也就是说，当你的应用程序中含有子类化自 QStyle 类的样式时，在 Qt 4.5 以前的版本 中就不要再同时设置样式表了。

9.4.1与 Qt Designer 集成使用

在 Qt Designer 中，可以很方便的设置样式表并浏览其效果。可以在 Qt Designer 设计 的界面之中的任意窗口部件上点击鼠标右键，在弹出的上下文菜单上选择 “改变样式表”，如 图 9-11 所示。



图 9-11 改变样式表的上下文菜单

接下来将弹出名为【编辑样式表】的对话框，如图 9-12 所示。其中有 4 个下拉按钮， 分别是添加资源、添加渐变、添加颜色和添加字体， 使用它们可以细化设置情况。



图 9-12 【编辑样式表】对话框

举个例子，依次点击添加颜色和添加字体按钮，为窗口部件设置颜色和字体的样式 。其 结果如图 9-13 所示。

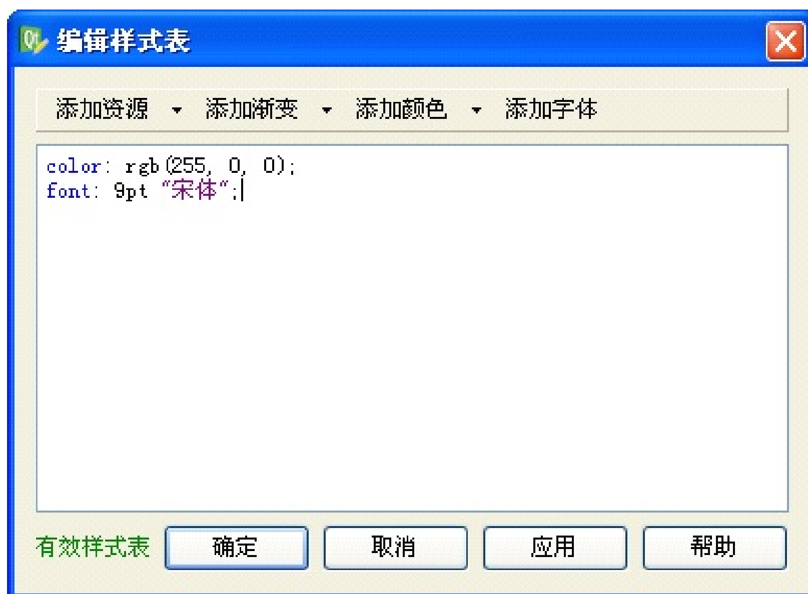


图 9-13 添加字体和设置颜色

从 Qt 4.2 开始， Qt Designer 内置了针对样式表的语法高亮和语法有效性校验支持 。 语法有效性指示器位于【编辑样式表】窗体的左下角。样式表的语法有效的话，将以绿色字 体指示出来“有效样式表”，如图 9-14 所示。



图 9-14 样式表设置正确

当你按下【确定】或者【应用】按钮，Qt Designer 将立即把基于新的样式表的程序外 观展现在面前。

语法不正确的话，将以红色字体指示出来“无效样式表”，如图 9-15 所示。



图 9-15 样式表的语法不正确

现在，我们举个具体的小例子，也算是一个练习题吧。打开 Qt Designer，在界面上放 置两个 Label、一个 LineEdit 和一个 ComboBox 窗口部件。通过使用右键菜单来改变各个窗 口部件的样式表，最终完成的界面情形如图 9-16 所示，大家可以自行尝试如何实现这个界面 布局，多试几遍就可以的。（提示：要把 LineEdit 设置成如图 9-16 所示的圆角形式，需要加 入语句 border-style:outset;）



图 9-16 一个具体的例子

9.4.2 实例：样式表的应用

下面我们以一个实例来讲解样式表的应用。这个例子取材于 Qt Demo，比较复杂，有一 定难度，基本上覆盖了前面几章讲述的各种技能点，主要包括：

- 如何自定义 Qt 的样式表
- 如何在应用程序中应用样式表
- 如何不使用样式表来设置应用程序的样式
- 如何使用单继承法从.ui 文件创建派生类
- 如何自定义资源集文件
- 如何使信号和槽自动连接
- 如何在两个窗口之间建立关联
- 元对象系统方法的使用

这个程序名字叫 stylesheet，其运行后的效果如图 9-17 所示。

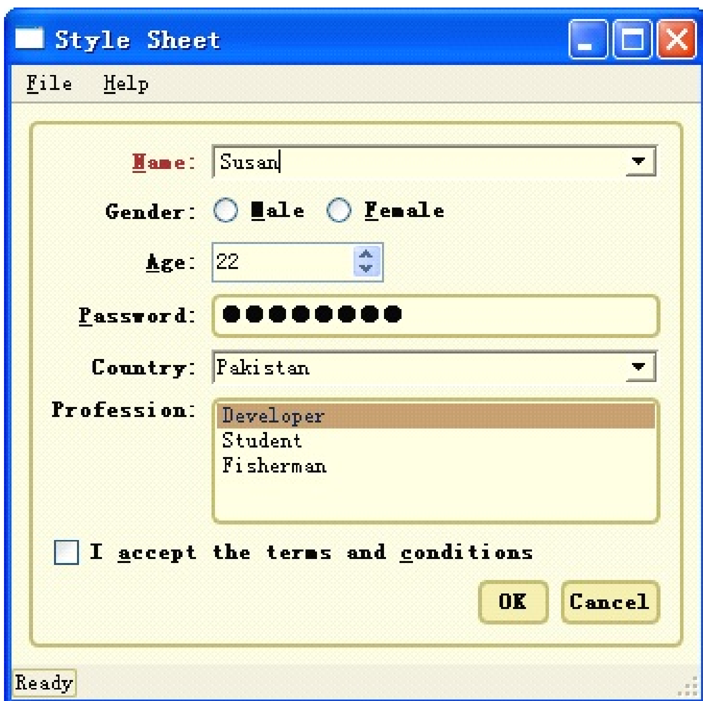


图 9-17 实例运行效果

该例子基于主窗口样式，有些类似于我们在网上所常见的填写个人资料的网站注册程序。

程序有两个主菜单，依次点击【File】->【Edit Style】菜单项，将弹出如图 9-18 所示的设置样式表的对话框，在其中内置了几种样式供选择，使用者也可以在编辑框中输入自 定义的样式。设置完成后，主界面的窗口部件的样式将依此相应的变化 。里面的 Coffee 样式自定义了 push button、frames 和 tooltip，但使用了下层的风格 （例如这里是 Windows XP风格）来绘制 checkbox，combobox 和 radio button。Pagefold 风格完全重新定义了对话框 中使用的所有控件的外观，从而实现了一种独特的，平台无关的外观。

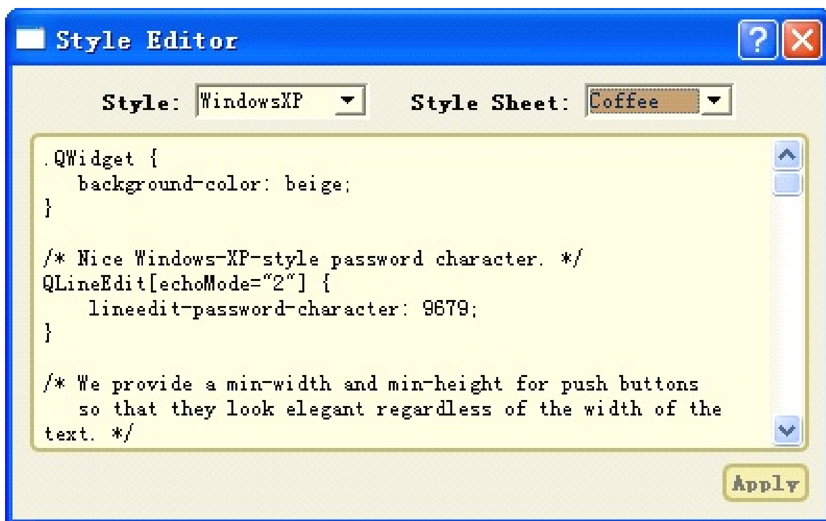


图 9-18 样式表编辑器

这个程序里面包含如下原生源文件：

```
mainwindow.ui
stylesheeteditor.ui
stylesheet.qrc
/qss/coffee.qss
/qss/default.qss
/qss/pagefold.qss
/images/*.png
mainwindow.h
mainwindow.cpp
stylesheeteditor.h
stylesheeteditor.cpp
```

其中，mainwindow.ui 和 stylesheeteditor.ui 分别是主程序和样式编辑器的界面布局 文件，是使用 Qt Designer 制作的。Stylesheet.qrc 是资源集文件，在其中描述了程序中用 到的样式表文件和图片文件的位置和名称 。在 qss 文件夹中包含了 3 个.qss 文件，它们描述 了程序中用到的样式，我们的程序将读取它们并转换成样式表 。在 images 文件夹中放置了程 序中用到的图片文件。mainwindow.h 和 mainwindow.cpp 构成了程序中的主程序类，而 stylesheeteditor.h 和 stylesheeteditor.cpp 构成了样式编辑器类。

下面我们就结合源代码为大家讲解这个程序的功能是怎样实现的。 首先看看 mainwindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H
#include &lt;QtGui&gt;
#include "ui_mainwindow.h"
class StyleSheetEditor;
class MainWindow : public QMainWindow
{
    Q_OBJECT
public:
    MainWindow();
private slots:
    void on_editStyleAction_triggered();
    void on_aboutAction_triggered();
private:
    StyleSheetEditor *styleSheetEditor;
    Ui::MainWindow ui;
};
#endif
```

第 1、2 和第 16 行一起构成了头文件的预定义卫哨，这样做的目的是为了防止程序中重复定义或包含头文件，是严谨的编程风格，建议大家遵循这一范例的做法。

第 3 行引入 QtGui 模块的声明，它包含了 QMainWindow 类的定义。 第 4 行引入 ui_mainwindow.h 的声明。

第 5 行采用前置声明的方式引入样式编辑器类 StyleSheetEditor。 第 6 行声明主程序类 MainWindow 单公有继承自 QMainWindow 类。 第 7 行时必需的，因为要用到 Qt 的核心机制，如信号/槽机制等。 第 8、9 行声明 MainWindow 类的构造函数。

第 10-12 行声明私有槽，它们的命名遵循了 Qt 信号/槽的“自动关联规则”。

第 13-15 行声明了私有成员，它们分别是样式编辑器类的对象和主程序界面类的对象 。 从这里也可以看出，我们对.ui 文件的引入将采用单继承的方式。

下面再来看一下 mainwindow.cpp 文件的内容。

```
#include "mainwindow.h"
#include "stylesheeteditor.h"
MainWindow::MainWindow()
{
    ui.setupUi(this);
    ui.nameLabel-&gt;setProperty("class", "mandatory QLabel");
    styleSheetEditor = new StyleSheetEditor(this);
    statusBar()-&gt;addWidget(new QLabel(tr("Ready")));
    connect(ui.exitAction, SIGNAL(triggered()), qApp, SLOT(quit()));
    connect(ui.aboutQtAction, SIGNAL(triggered()), qApp, SLOT(aboutQt()));
}
void MainWindow::on_editStyleAction_triggered()
{
    styleSheetEditor-&gt;show();
    styleSheetEditor-&gt;activateWindow();
}
void MainWindow::on_aboutAction_triggered()
{
    QMessageBox::about(this, tr("About Style sheet"),
    tr("The &lt;b&gt;Style Sheet&lt;/b&gt; example shows how widgets can be styled "
    "using &lt;a href=\"http://doc.trolltech.com/4.5/stylesheet.html\"&gt;Qt "
    "Style Sheets&lt;/a&gt;. Click &lt;b&gt;File|Edit Style Sheet&lt;/b&gt; to pop up the
    "style editor, and either choose an existing style sheet or design "
    "your own."));
}
```

第 1、2 两行引入程序中用到的头文件声明。

第 4 行初始化界面布局，setupUi()函数一般要放在构造函数内的第一行。 第 5 行设置 nameLabel 的属性，setProperty()方法的原型如下：

```
bool QObject::setProperty ( const char * name, const QVariant & value )
```

它将对象的 name 的属性值设置为 value。对于本句来说，就是把 nameLabel 的 class 属性值设置为"mandatory QLabel"，这将使得该窗口部件被突出显示。

第 6 行实例化 styleSheetEditor 类的对象，其父窗口为 MainWindow。

第 7 行为状态栏添加一个窗口部件，显示 "Ready"字样，即汉语的“就绪“。

第 8、9 行分别连接菜单项 exitAction 和 aboutQtAction 的触发信号 triggered()和全局对象 qApp 的 quit()槽和 aboutQt()槽。

第 10-12 行定义了 on_editStyleAction_triggered()槽函数。 第 11 行显示样式编辑器窗口。

第 12 行调用 activateWindow()方法把样式编辑器窗口设置为活动窗口。 activateWindow()方法的原型如下：

```
void QWidget::activateWindow ()
```

它用来把某个顶层窗口设置为当前的活动窗口。

小贴士：活动窗口与如何设置活动窗口 所谓活动窗口就是当前能够拥有键盘输入焦点的，可见的顶层窗口。 activateWindow()方法的作用与用鼠标单击顶层窗口的标题栏的效果是一样的。在 X11上，这个效果并不确定，因为它依赖于具体的窗口管理器，如果要确保某窗口为活动窗口，

最好是在调用 activateWindow()方法之后再调用 raise()方法，后者本身也是 Qt 内置的一个 槽函数。但无论如何，在这之前都需要确保该窗口首先是可见的，否则将没有任何效果。

在 Windows 平台上，这种情形又有些差异。因为 Microsoft 不允许一个应用打断由另一 个应用建立起来的和用户的对话（即交互过程 ），所以 activateWindow()被调用后，该窗口 部件可能还是不能成为顶层的活动窗口，而只是它的标题栏变为高亮，以告诉用户该窗口部 件的状态发生了某些变化。这一点，请读者朋友在使用时注意。

如果想知道某窗口是否为活动窗口，可以调用 isActiveWindow()方法。 前面我们曾经讲到过 Qt 核心机制中的“资源集文件“。现在我们看一下如何在 Qt Creator 中书写或配置资源集文件。

在 Qt Creator 中用鼠标双击资源集文件，将打开资源集文件编辑器，在资源集编辑器中可以增加或删除文件、节点等，其操作比较方便，一目了然。如图 9-19 所示。

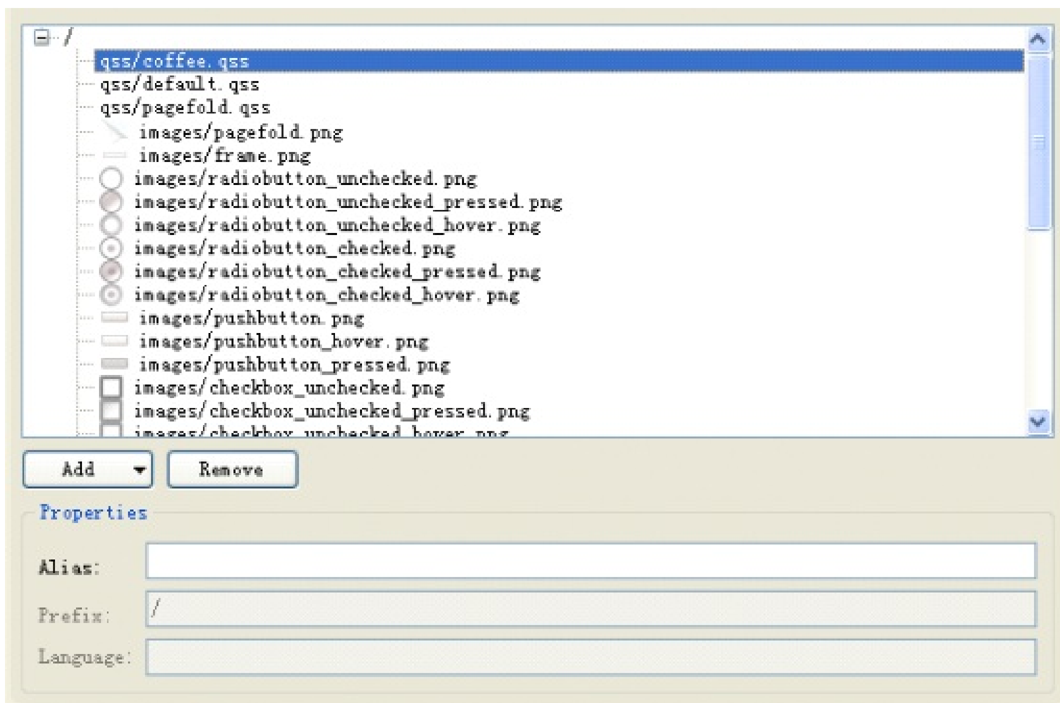


图 9-19 资源集文件编辑器

我们再来看一下 stylesheeteditor.h 文件的内容。

```
#ifndef STYLESHEETEDITOR_H
#define STYLESHEETEDITOR_H
#include &lt;QDialog&gt;
#include "ui_stylesheeteditor.h"
class StyleSheetEditor : public QDialog
{
    Q_OBJECT
public:
    StyleSheetEditor(QWidget *parent = 0);
private slots:
    void on_styleCombo_activated(const QString &styleName);
    void on_styleSheetCombo_activated(const QString &styleSheetName);
    void on_styleTextEdit_textChanged();
    void on_applyButton_clicked();
private:
    void loadStyleSheet(const QString &sheetName);
    Ui::StyleSheetEditor ui;
};
#endif
```

第 1 行声明 StyleSheetEditor 类单公有继承自 QDialog。

第 5-9 行声明了程序中的私有槽，命名遵循了 Qt 信号/槽的“自动关联规则”。

第 11 行声明了私有方法 loadStyleSheet()，它用来从.qss 文件中读取内容并转化问样式表。第 12 行声明了私有成员变量 ui。

再来看一下 stylesheeteditor.cpp。

```
#include &lt;QtGui&gt;
#include "stylesheeteditor.h"
StyleSheetEditor::StyleSheetEditor(QWidget *parent)
: QDialog(parent)
{
    ui.setupUi(this);
    QRegExp regExp(".(.*)\\+?Style");
    QString defaultStyle = QApplication::style()-&gt;metaObject()-&gt;className();
    if (regExp.exactMatch(defaultStyle))
    {
        defaultStyle = regExp.cap(1);
    }
    ui.styleCombo-&gt;addItems(QStyleFactory::keys());
    ui.styleCombo-&gt;setCurrentIndex(ui.styleCombo-&gt;findText(defaultStyle,
    Qt::MatchContains));
    ui.styleSheetCombo-&gt;setCurrentIndex(ui.styleSheetCombo-&gt;findText("Coffee"));
loadStyleSheet("Coffee");
}
void StyleSheetEditor::on_styleCombo_activated(const QString &styleName)
{
    qApp-&gt;setStyle(styleName);
    ui.applyButton-&gt;setEnabled(false);
}
void StyleSheetEditor::on_styleSheetCombo_activated(const QString &sheetName)
{
    loadStyleSheet(sheetName);
}
void StyleSheetEditor::on_styleTextEdit_textChanged()
{
    ui.applyButton-&gt;setEnabled(true);
}
void StyleSheetEditor::on_applyButton_clicked()
{
    qApp-&gt;setStyleSheet(ui.styleTextEdit-&gt;toPlainText());
    ui.applyButton-&gt;setEnabled(false);
}
void StyleSheetEditor::loadStyleSheet(const QString &sheetName)
{
    QFile file(":/qss/" + sheetName.toLower() + ".qss");
    file.open(QFile::ReadOnly);
    QString styleSheet = QLatin1String(file.readAll());
    ui.styleTextEdit-&gt;setPlainText(styleSheet);
    qApp-&gt;setStyleSheet(styleSheet);
    ui.applyButton-&gt;setEnabled(false);
}
```

第 3-6 行调用 Qt 中的正则表达式来取得默认的样式。

第 3 行定义一个正则表达式变量 regExp。

第 4 行是 Qt 元对象机制（Meta-Object System）的应用。

首先通过 style()方法获得应用程序的样式，然后利用 Qt 元对象机制获取元对象信息。 style()方法是 QApplication 类的静态方法，其原型如下：

```
QStyle * QApplication::style () [static]
```

它将返回应用程序的样式对象。

这里简单的讲解一下 Qt 元对象机制，详细的内容请见第 13 章。

Qt 的元对象系统是 Qt 的核心机制之一，Qt 的信号/槽机制、属性系统、运行时型别信息 等机制都是以元对象系统为基础的。在应用程序中，每一个 QObject 的子类都会有一个单独的 Qt 元对 象实例， 这个实例 中保存了 该类所有 的元对象 信息，该 实例可以 通过调用 QObject::metaObject()方法来得到。

而 QMetaObject 被称作是 Qt 的元对象类，它包含了 Qt 对象的元信息。

QObject::metaObject()方法的原型如下：

```
const QMetaObject * QObject::metaObject () const [virtual]
```

可以看到，它返回本对象的指向其元对象的指针。 用来获取元对象信息的方法有很多，本程序中用到的 className()是其中的一种，它返回类 的名字。

第 5、6 两行应用正则表达式的校验来获得 defaultStyle 的值。

第 8 行是为 styleCombo 设置当前下拉框列表中的默认选项。 findText()是 QcomboBox 类的方法，其原型如下：

```
int QComboBox::findText ( const QString & text, Qt::MatchFlags flags =
Qt::MatchExactly | Qt::MatchCaseSensitive ) const
```

它在 QComboBox 对象的 item 列表中搜索与 text 相匹配的项的索引值，如果没有相匹配的项，则返回-1，该值是 int 型的。flags 参数给出了搜索的方法，即如何与 text 相匹配。

flags 的值取自枚举值 Qt::MatchFlag，后者描述了在一个模型中搜索时可以遵循的匹配原则，如表 9-2 所示。

表 9-2 MatchFlag 的取值

常量	值	说明
Qt::MatchExactly	0	执行 QVariant 匹配（QVariant 可以看做是 Qt 的最常用变量类型的联合体）
Qt::MatchFixedString	8	执行按字符匹配。注意这种方式默认情况下不区分大小写，只有同时指定 Qt::MatchCaseSensitive 才区分大小写。
Qt::MatchContains	1	搜索条件包含在（QComboBox 的下拉列表）项目中
Qt::MatchStartsWith	2	匹配条件是与项的开头相匹配，即“以 XXX 开头”
Qt::MatchEndsWith	3	搜索的条件与项的结尾相匹配，即“以 XXX 为结尾”
Qt::MatchCaseSensitive	16	执行大小写敏感匹配搜索
Qt::MatchRegExp	4	以一个正则表达式为匹配条件执行按字符匹配搜素，注意这种方式不区分大 小写，除非同时指定 Qt::MatchCaseSensitive 条件。
Qt::MatchWildcard	5	
Qt::MatchWrap	32	执行类似“令牌环”式的搜索，对每一个项都要经过验证
Qt::MatchRecursive	64	执行递归搜索

所有上述这些标记符号都是 QFlags< MatchFlag >的一部分，它们之间的组合可以用 OR 来连接，在程序中即是使用|符号。这些标记符号经常与 QRegExp 类结合使用。

第 9 行与第 8 行同理，设置 styleSheetCombo 的当前项为 Coffee，也即当前的样式表的 名称为 Coffee。

紧接着，第 10 行调用 loadStyleSheet()方法相应的设置应用程序的样式表为 Coffee。 第 21-27 行是 loadStyleSheet()方法的定义。

第 22 行使用 QFile 类的对象根据实参取得 Coffee 样式表对应的.qss 文件全名，即带路 径的 Coffee.qss。

第 23 行以只读方式打开 Coffee.qss 文件。

第 24 行读取文件的全部内容。QLatin1String 类的构造函数原型如下：

```
QLatin1String::QLatin1String ( const char * str )
```

注意它的参数是 const char * str，而不是 QString 类的对象。

小贴士：QLatin1String 类的使用

关于 QLatin1String 类的使用有很多话题。这里只讲一下最为基本的部分。 QLatin1String 类为采用 ASCII/Latin-1 编码形式的字符串操作提供了一个轻量级的封装。

通常在操作字符串时，Qt 推荐开发者尽量不要直接使用 QString 类，因为它的速度比较慢。替代的方法一种是使用 const char，使用它可以避免创建一个临时的 *QString* 对象， 这样就节省了开销。很多 *QString* 的成员函数如 *insert()*、*replace()*、*index()*等都接受 *const char*作为参数或返回值。举个例子，在下面的所有的示例代码中假设变量 str 是 QString 类型。

```
if (str == "auto" || str == "extern"
    || str == "static" || str == "register")
{
    ...
}
```

上面和下面的两段代码实现了相同的操作 。由于下面的代码在执行期间创建了 4 个临时的 QString 变量并且对字串值进行了深度拷贝，增大了程序运行的开销，所以上面这段代码 的执行速度就要明显的快于下面这段代码。

```
if (str == QString("auto") || str == QString("extern")
    || str == QString("static") || str == QString("register"))
{
    ...
}
```

如果使用 QLatin1String 类，代码可以写成下面这样，书写起来比较费力气些，但是代码执行的效率很高，比使用 QString::fromLatin1()要快得多。

```
if (str == QLatin1String("auto")
    || str == QLatin1String("extern")
    || str == QLatin1String("static")
    || str == QLatin1String("register")
{
    ...
}
```

基本上在各种场合都可以使用 QLatin1String 来代替 QString，就像下面代码中示例的这种用法。

```
QLabel *label = new QLabel(QLatin1String("MOD"), this);
```

第 25 行使用 setPlainText()方法把 styleTextEdit 要显示的内容设置为样式表文件的内容。

第 26 行设置应用程序的样式表。 代码段中的其它部分比较简单，不再赘述了。

再来看一看 main.cpp 的内容。

```
#include &lt;QtGui&gt;
#include "mainwindow.h"
int main(int argc, char *argv[])
{
    Q_INIT_RESOURCE(stylesheet);
    QApplication app(argc, argv);
    MainWindow window;
    window.show();
    return app.exec();
}
```

这里第 1 行最为重要，要使用资源集文件，就要使用 Q_INIT_RESOURCE 宏。

小贴士：Q_INIT_RESOURCE 宏的使用 Q_INIT_RESOURCE 宏的原型如下：

```
void Q_INIT_RESOURCE ( name )
```

该宏的作用是初始化在 name 指定的.qrc 文件中的资源文件。通常，Qt 在应用程序初始化时自动加载资源。注意，在某些平台上使用静态链接库时，必须使用 Q_INIT_RESOURCE() 宏来存储资源。

举个例子，如果应用程序中用到的资源文件列在名为 myapp.qrc 的资源集文件中，那么 为了确保资源在应用程序初始化时被加载，你需要确保在主程序的 main()中加上下面这句：

```
Q_INIT_RESOURCE(myapp);
```

有两点需要注意，一是参数 name 的命名必须符合标准 C++对于变量命名的规范，不可以含有不合适的字符。

第二，该宏不能用在名字空间里面。它必须在 main()主函数中被调用。如果必须在名字 空间中使用，可以使用下面的示例代码。

```
inline void initMyResource() { Q_INIT_RESOURCE(myapp); }
namespace MyNamespace
{
    ...
    void myFunction()
    {
        initMyResource();
    }
}
```

项目文件 stylesheet.pro 可以像下面这样书写。

```
TEMPLATE = app
TARGET =
DEPENDPATH += .
INCLUDEPATH += .
# Input
HEADERS += mainwindow.h stylesheeteditor.h
FORMS += mainwindow.ui stylesheeteditor.ui
SOURCES += main.cpp mainwindow.cpp stylesheeteditor.cpp
RESOURCES += stylesheet.qrc
```

第 1 行表明应用程序的模板为 app。第 5 行是注释，第 9 行表明要包含用到的资源集文件。

通过本实例，大家可以看到，样式表是一种在运行时解释的普通文本文件，使用它们不

需要具备编程知识。我们可以在应用程序级别和窗口级别设置样式表。另外，如果在不同的级别都设置了样式表，则应用程序将继承所有有效的样式，你看到的程序样子是这些效果的叠加，这被称作是样式表效果的层叠（ cascading）。

9.5 问题与解答

问：CSS 样式表的命令是否不能完全适用于 Qt 中？

答：在 Qt 中，样式表文件可以存成.qss 文件。QSS 在设计时参考了 CSS 的设计，但两 者在语法、属性等方面有些差别，所以 CSS 不能完全适用于 Qt。

问：在 Qt 中如何读取 CSS 样式表中的某个属性的具体值？ 答：在本章里面已经介绍过，样式表是一种在运行时解释的普通文本文件 ，通常可以保存成.qss 文件。可以使用 QFile 类读取、设置其中的内容，具体请参考本章的实例。

问：用 Qt Designer 的样式表添加图片和用 drawPixmap()函数实现显示图片哪个效果 更好？

答：drawPixmap()函数使用了缓冲技术，消耗的系统资源比较少。通常情况下，使用样 式表不如直接调用 drawPixmap()函数来绘图效率高。尤其是在嵌入式系统中，在系统资源有 限的情况下，尽量不要使用样式表。但是 样式表也有优势，就是它易于控制和使用，在系统 资源比较丰富的情况下，比如桌面环境，也可酌情使用。

问：如何给应用程序中的某一个窗口部件单独设置样式表 ，我怎么设置都没有效果？这 是 QT 帮助文档里自带的说将 stylesheet 应用到具体对象上的方法，但似乎不起作用。

```
ID Selector
QPushButton#okButton
Matches all QPushButton instances whose object name is okButton.
```

答： 会起作用的，可能文档中的一个关键点你没有注意到，请看下面的示例

```
QDialog myDialog;
myDialog.setObjectName("mydialog");//如果要对这个对象单独设 stylesheet 的话一定要设置它的
objectName
```

然后再像下面这样写就可以了：

```
QDialog#mydialog {background-image: url(2.bmp);
```

9.6 总结与提高

本章主要讲述了以下内容，希望大家能够熟练掌握。

- 什么是应用程序的观感
- 什么是 Qt 样式表
- 样式表的作用
- 样式表的基本语法
- 样式表与 Qt Designer 的结合使用
- 使用样式表设置应用程序或其子部件的观感
- 综合使用 QStyle 类和 Qt 样式表的方法

由于 Qt 样式表的引入，定制 Qt 部件的外观样式变得非常简单。 无论你是想仅仅修改 一个现有部件的外观，还是想从零开始设计一套全新的界面风格， 现在都有了一种新的方法 而不必再去继承并实现一个 QStyle 的子类。使用 Qt 的样式表，再结合布局管理的相关技巧， 我们就可以随心所欲的设计应用程序的用户界面了。

第 10 章 在程序中使用.ui 文件

本章重点

- 了解 uic 的使用方法
- 了解 Ui_YourFormName.h 文件的组成
- 熟练掌握在编译时加入处理.ui 文件的方法，包括直接使用法、单继承法和多继承法
- 掌握在运行是加入处理.ui 文件的方法，主要是动态加载的方法
- 熟练掌握信号与槽实现自动关联的方法

当应用程序被构建时，使用 Qt 提供的构建工具，如 qmake 和 uic，可以把使用 Qt Designer 创建的用户界面（.ui 文件）集成进工程里面，并自动生成相对应的文件和代码。Qt 提供了两种引入和处理.ui 文件的方法：一种是在编译时引入处理（Compile Time Form Processing），一种是在运行时引入处理（Run Time Form Processing）。前者又可以有 3 种做法：直接使用 法、单继承法和多继承法；而后者是动态的加载 .ui 文件，这需要借助 QtUiTools 模块来完 成。

首先让我们先来了解一下 Qt 提供的的 uic 工具。

10.1 uic 的使用

uic 的全称是 User Interface Compiler for the Qt GUI toolkit，顾名思义，它就 是.ui 文件编译器，也是 Qt 提供的工具之一。它的主要功用可以用一句话来概括，就是读取 由 Qt Designer 制作的用户界面文件（即.ui 文件），并生成相对应的 C++头文件。该头文件 的形式是 Ui_YourFormName.h。

使用方法如下：

```
uic [options] &lt;uifile&gt;
```

对应到我们的实例即可写成：

```
uic [options] YourFormName.ui
```

其中，[options]选项内容如表 10-1 所示

表 10-1 uic 命令行选项

Option	描 述
-o <file>	输出到<file>中，而不是采用标准输出，其中 file 为文件名
-tr <func>	使用<func> 来翻译字符串，以代替 tr()的使用
-p	不产生重复包含的卫哨
-h	显示 options 列表和使用方法，相当于 uic 的帮助
-v	显示 uic 的版本号

小贴士：现在我们一般不直接使用 uic，当使用 qmake 时，uic 会在需要时被 qmake 自动调用。

10.2 Ui_YourFormName.h 文件的组成

我们了解了 uic 工具的使用方法和公用，现在来看看它为我们生成的 C++头文件的组成 是什么样的，里面的“构件”又有哪些功用，这对我们理解后面几节的内容有着重要的作用 。

我们将使用 Qt Designer 创建一个界面文件，名为 calculatorform.ui，这个程序主要 是完成简单的加法计算功能。该界面的元素组合如图 10-1 所示。

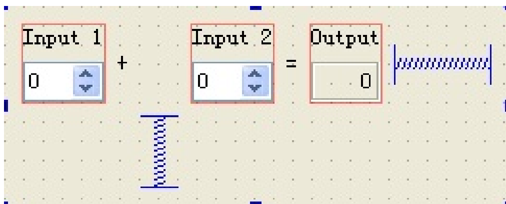


图 10-1 使用 Qt Designer 制作的界面

怎样做到在不使用 qmake 的情况下，就可以产生对应的头文件呢，我们遵循上一节介绍 的方法，在 Qt 的命令行下面输入如下命令：

```
uic -o ui_calculatorform.h calculatorform.ui
```

小贴士：再次提醒，以 Windows XP 操作系统为例，进入 Qt 命令行的方法是：依次点击【开始】->【所有程序】->【Qt SDK by Nokia v2009.03 (open source)】->【Qt Command Prompt】。然后切换到你的.ui 文件所在的目录，执行上述命令即可。

以下是据此生成的 ui_calculatorform.h 文件的完整内容，我将分段向大家讲解。

```
/********************************************************************************
** Form generated from reading ui file 'calculatorform.ui'
**
** Created: Sun Sep 6 22:34:26 2009
** by: Qt User Interface Compiler version 4.5.2
**
** WARNING! All changes made in this file will be lost when recompiling ui file!
********************************************************************************/
```

这一段代码是 uic 工具自动生成的注释内容，它记录了该 C++头文件对应的原始.ui 文件的名字以及产生的时间和使用的 uic 的版本。最后一行提示内容很重要，它明确的告诉我们，不要手动修改该文件内容，因为在下次运行 qmake 或者 uic 时，它们会被覆盖掉。所以，我们如果需要对文件内容做修改，那么到 Qt Designer 中修改界面元素就行了，uic 会把所 有的改动反映出来。

```
#ifndef UI_CALCULATORFORM_H
#define UI_CALCULATORFORM_H
#include &lt;QtCore/QVariant&gt;
#include &lt;QtGui/QAction&gt;
#include &lt;QtGui/QApplication&gt;
#include &lt;QtGui/QButtonGroup&gt;
#include &lt;QtGui/QGridLayout&gt;
#include &lt;QtGui/QHeaderView&gt;
#include &lt;QtGui/QLabel&gt;
#include &lt;QtGui/QSpacerItem&gt;
#include &lt;QtGui/QSpinBox&gt;
#include &lt;QtGui/QVBoxLayout&gt;
#include &lt;QtGui/QWidget&gt;
```

上面这段的开头两句加上该文件结尾的 #endif // UI_CALCULATORFORM_H 这句构成了完整的防止头文件重复定义的卫哨，这是很好的编程规范，我们在自己书写代码时，也要遵循 这种严谨的做法。

后面的几句加入了程序中用到的窗口部件以及变量的头文件 。请大家注意它们的用法也 是非常规范的，正是许多 C++大师所提倡的“用到什么就包含什么，不用的就不包含 ”的做 法的应用典范。笔者发现有很多的朋友不论什么情况，都喜欢加入一句 #include <QtGui>或 者 #include <QtCore>，这固然省事，因为它们包含了这两个模块下的所有子模块的定义 ， 但是这样一来，就降低了程序的编译速度，使得性能下降。

```
QT_BEGIN_NAMESPACE
class Ui_CalculatorForm
{
public:
QGridLayout *gridLayout;
QSpacerItem *spacerItem;
QLabel *label_3_2;
QVBoxLayout *vboxLayout;
QLabel *label_2_2_2;
QLabel *outputWidget;
QSpacerItem *spacerItem1;
QVBoxLayout *vboxLayout1;
QLabel *label_2;
QSpinBox *inputSpinBox2;
QLabel *label_3;
QVBoxLayout *vboxLayout2;
QLabel *label;
QSpinBox *inputSpinBox1;
```

上面这段代码中，首先用 QT_BEGIN_NAMESPACE 宏表示开始进入名字空间。然后定义了一个名为 Ui_CalculatorForm 的类，它实际上是界面的实体类，界面上的所有元素都被定义为相应的窗口部件的实例，并且它们一定是被定义为 public,即公有的成员，这将使得后面的 Ui 名字空间内的派生类能够继承它们。

```
void setupUi(QWidget *CalculatorForm)
{
    if (CalculatorForm-&gt;objectName().isEmpty())
        CalculatorForm-&gt;setObjectName(QString::fromUtf8("CalculatorForm"));
    CalculatorForm-&gt;resize(400, 300);
    QSizePolicy sizePolicy(static_cast&lt;QSizePolicy::Policy&gt;(5),
    static_cast&lt;QSizePolicy::Policy&gt;(5));
    sizePolicy.setHorizontalStretch(0);
    sizePolicy.setVerticalStretch(0);
```

```
    sizePolicy.setHeightForWidth(CalculatorForm-&gt;sizePolicy().hasHeightForWidth());
    CalculatorForm-&gt;setSizePolicy(sizePolicy);
    gridLayout = new QGridLayout(CalculatorForm);
    #ifndef Q_OS_MAC
    gridLayout-&gt;setSpacing(6);
    #endif
    #ifndef Q_OS_MAC
    gridLayout-&gt;setMargin(9);
    #endif
    gridLayout-&gt;setObjectName(QString::fromUtf8("gridLayout"));
    gridLayout-&gt;setObjectName(QString::fromUtf8(""));
    spacerItem = new QSpacerItem(40, 20, QSizePolicy::Expanding, QSizePolicy::Minimum);
    gridLayout-&gt;addItem(spacerItem, 0, 6, 1, 1);
    label_3_2 = new QLabel(CalculatorForm);
    label_3_2-&gt;setObjectName(QString::fromUtf8("label_3_2"));
    label_3_2-&gt;setGeometry(QRect(169, 9, 20, 52));
    label_3_2-&gt;setAlignment(Qt::AlignCenter);
    gridLayout-&gt;addWidget(label_3_2, 0, 4, 1, 1);
    vboxLayout = new QVBoxLayout();
    #ifndef Q_OS_MAC
    vboxLayout-&gt;setSpacing(6);
    #endif
    vboxLayout-&gt;setMargin(1);
    vboxLayout-&gt;setObjectName(QString::fromUtf8("vboxLayout"));
    vboxLayout-&gt;setObjectName(QString::fromUtf8(""));
    label_2_2_2 = new QLabel(CalculatorForm);
    label_2_2_2-&gt;setObjectName(QString::fromUtf8("label_2_2_2"));
    label_2_2_2-&gt;setGeometry(QRect(1, 1, 36, 17));
    vboxLayout-&gt;addWidget(label_2_2_2);
    outputWidget = new QLabel(CalculatorForm);
    outputWidget-&gt;setObjectName(QString::fromUtf8("outputWidget"));
    outputWidget-&gt;setGeometry(QRect(1, 24, 36, 27));
    outputWidget-&gt;setFrameShape(QFrame::Box);
    outputWidget-&gt;setFrameShadow(QFrame::Sunken);
    outputWidget-&gt;setAlignment(Qt::AlignAbsolute|Qt::AlignBottom|Qt::AlignCenter|Qt::A
    |Qt::AlignHorizontal_Mask|Qt::AlignJustify|Qt::AlignLeading|Qt::AlignLeft|Qt::AlignRi
    AlignTop|Qt::AlignTrailing|Qt::AlignVCenter|Qt::AlignVertical_Mask);
    vboxLayout-&gt;addWidget(outputWidget);
    gridLayout-&gt;addLayout(vboxLayout, 0, 5, 1, 1);
    spacerItem1 = new QSpacerItem(20, 40, QSizePolicy::Minimum, QSizePolicy::Expanding);
    gridLayout-&gt;addItem(spacerItem1, 1, 2, 1, 1);
    vboxLayout1 = new QVBoxLayout();
    #ifndef Q_OS_MAC
    vboxLayout1-&gt;setSpacing(6);
    #endif
    vboxLayout1-&gt;setMargin(1);
    vboxLayout1-&gt;setObjectName(QString::fromUtf8("vboxLayout1"));
    vboxLayout1-&gt;setObjectName(QString::fromUtf8(""));
    label_2 = new QLabel(CalculatorForm);
    label_2-&gt;setObjectName(QString::fromUtf8("label_2"));
    label_2-&gt;setGeometry(QRect(1, 1, 46, 19));
    vboxLayout1-&gt;addWidget(label_2);
    inputSpinBox2 = new QSpinBox(CalculatorForm);
    inputSpinBox2-&gt;setObjectName(QString::fromUtf8("inputSpinBox2"));
    inputSpinBox2-&gt;setGeometry(QRect(1, 26, 46, 25));
    vboxLayout1-&gt;addWidget(inputSpinBox2);
    gridLayout-&gt;addLayout(vboxLayout1, 0, 3, 1, 1);
    label_3 = new QLabel(CalculatorForm);
    label_3-&gt;setObjectName(QString::fromUtf8("label_3"));
    label_3-&gt;setGeometry(QRect(63, 9, 20, 52));
    label_3-&gt;setAlignment(Qt::AlignCenter);
    gridLayout-&gt;addWidget(label_3, 0, 1, 1, 1);
    vboxLayout2 = new QVBoxLayout();
    #ifndef Q_OS_MAC
    vboxLayout2-&gt;setSpacing(6);
    #endif
    vboxLayout2-&gt;setMargin(1);
    vboxLayout2-&gt;setObjectName(QString::fromUtf8("vboxLayout2"));
    vboxLayout2-&gt;setObjectName(QString::fromUtf8(""));
    label = new QLabel(CalculatorForm);
    label-&gt;setObjectName(QString::fromUtf8("label"));
    label-&gt;setGeometry(QRect(1, 1, 46, 19));
```

```
    vboxLayout2-&gt;addWidget(label);
    inputSpinBox1 = new QSpinBox(CalculatorForm);
    inputSpinBox1-&gt;setObjectName(QString::fromUtf8("inputSpinBox1"));
    inputSpinBox1-&gt;setGeometry(QRect(1, 26, 46, 25));
    vboxLayout2-&gt;addWidget(inputSpinBox1);
    gridLayout-&gt;addLayout(vboxLayout2, 0, 0, 1, 1);
    retranslateUi(CalculatorForm);
    QMetaObject::connectSlotsByName(CalculatorForm);
} // setupUi
```

上面这段代码是界面实体类的公有方法 setupUi()的内容。这是最为重要的一个成员函数，它完成了整个界面的构造和布局，代码的内容比较简单，遵循了我们前面反复讲到的使 用 Qt 编程的基本步骤，即先声明要使用到的窗口部件，然后实例化它们，再为它们设置属 性和方法。

QMetaObject::connectSlotsByName(CalculatorForm);这一句是需要重点说明的，因为 它实现了 CalculatorForm 界面实体类中的信号与槽机制。这个方法的原型是：

```
void QMetaObject::connectSlotsByName ( QObject * object ) [static]
```

它是 QMetaObject 类的静态方法，作用是递归搜索 object 对象及子对象，然后将其中相匹配的信号与槽连接起来。所谓相匹配的信号与槽，就是要求槽函数的书写形式如下所示：

```
void on_&lt;object name&gt;_&lt;signal name&gt;(&lt;signal parameters&gt;);
```

从编程人员的角度来看，只要将槽函数遵循上面的书写形式，就能够实现信号和槽的自动关联，而无需手写代码显式的完成。

再举个例子，假设我们的界面实体类中含有一个子对象，它的类型是 QPushButton，它 的 objectName 属性被设置为 btn1。那么为了使槽函数与信号 clicked()能够自动关联，我们 必须将槽函数写成如下形式：

```
void on_button1_clicked();
```

小贴士：QMetaObject 类是 Qt 元对象系统（Meta-Object System）的组成部分，是实现信号/槽机制、运行时类型信息以及 Qt 属性系统的基础。在一个应用程序中所有的 QObject 子类 都会创建一个属于自己的单独的 QMetaObject 的实例，并且这个实例存储了该子类的所有元 对象信息。这些通常包括类名（class name）、超类名（superclass name）、属性信息（properties）、信号与槽（signals and slots）等。

关于 Qt 的元对象系统，我们会在后面的第 13 章作详细介绍。

在后面我们会介绍编译时处理.ui 文件的 3 种方法：直接使用法、单继承法和多继承法。 无论使用它们中的哪一种，都要显式的调用 setupUi()这个方法来完成界面元素的构造和布 局，而它的参数就是你自定义的窗体类，这一点请大家在阅读时注意。

```
void retranslateUi(QWidget *CalculatorForm)
{
    CalculatorForm-&gt;setWindowTitle(QApplication::translate("CalculatorForm",
        "Calculator Form", 0, QApplication::UnicodeUTF8));
    label_3_2-&gt;setText(QApplication::translate("CalculatorForm", "=", 0,
        QApplication::UnicodeUTF8));
    label_2_2_2-&gt;setText(QApplication::translate("CalculatorForm", "Output", 0,
        QApplication::UnicodeUTF8));
    outputWidget-&gt;setText(QApplication::translate("CalculatorForm", "0", 0,
        QApplication::UnicodeUTF8));
    label_2-&gt;setText(QApplication::translate("CalculatorForm", "Input 2", 0,
        QApplication::UnicodeUTF8));
    label_3-&gt;setText(QApplication::translate("CalculatorForm", "+", 0,
        QApplication::UnicodeUTF8));
    label-&gt;setText(QApplication::translate("CalculatorForm", "Input 1", 0,
        QApplication::UnicodeUTF8));
    Q_UNUSED(CalculatorForm);
} // retranslateUi
```

以上这段是 retranslateUi 方法的定义，它使得应用程序具备了支持国际化的能力 。其 中调用了 translate()方法，它是在 Qt 4.5 以后新引入的。

```
namespace Ui {
    class CalculatorForm: public Ui_CalculatorForm {};
} // namespace Ui
QT_END_NAMESPACE
#endif // UI_CALCULATORFORM_H
```

上面这段代码定义了名字空间 Ui，在其内部声明了一个名为 CalculatorForm 的类，它单公有继承自界面实体类 Ui_CalculatorForm，这就使得它继承了界面实体类的所有公有成 员。声明名字空间的用意是防止混淆命名。

注意，这个类的名字是与你的.ui 文件中 Form 的 objectName 的属性值相一致的。在后面的章节里，我们都要与这个类打交道，而不与界面实体类 Ui_CalculatorForm 发生联系，这样做就使得应用程序的界面设计与代码尽可能的分离开来，是现代编程思想的体现。

10.3 编译时加入处理.ui 文件的方法

编译时加入处理.ui 文件通常可以采用 3 种方式：直接使用法、单继承法和多继承法。 下面我们就以 Calculator Form 这个程序为例，分别介绍这 3 种方式的使用。 Calculator Form 程序主要实现了简单的加法计算功能，在 Qt Designer 中绘制的用户界面如图 10-2 所示。

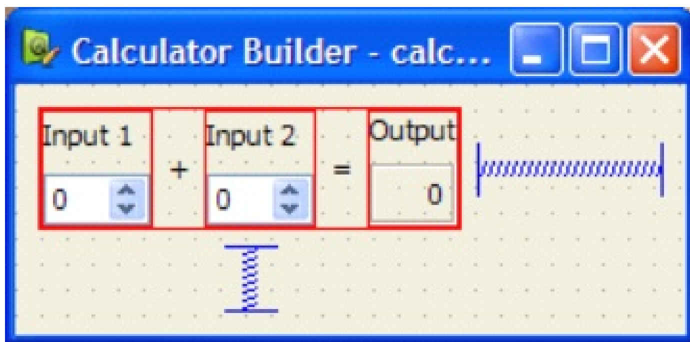


图 10-2 Calculator Form 的.ui 文件

10.3.1 直接使用法

本小节例子的代码是来自源码包中的 calculatorform-direct 实例。

1.使用要点

这种方法的要领是创建一个子类化 QWidget 的自定义类，把它作为“容器”来存放在 calculatorform.ui 中描绘的窗体。

2.实例讲解

我们的应用程序工程中通常包含一个 main.cpp 文件以及一个.ui 文件。 如果全部采用手写代码的话，我们需要在 qmake 工程文件.pro 中加入几行：

```
TEMPLATE = app
FORMS = calculatorform.ui
SOURCES = main.cpp
```

第 2 句告诉 qmake 使用 uic 编译 calculatorform.ui 文件，这将生成

ui_calculatorform.ui 文件,通过前面对该文件内容的分析，我们知道这个文件中有一个界 面实体类，其中包含所有的界面元素的描述和配置情况， 并且 qmake 为我们生成了一个单公有继承自这个界面实体类的一个子类 - CalculatorForm，它位于 Ui 名字空间中。

qmake 将据此生成 ui_calculatorform.h 文件，我们需要在 SOURCES 变量中列出的文件 加入这个文件的声明（这里只有 main.cpp 文件）：

```
#include "ui_calculatorform.h"
```

在 main()函数中我们创建了一个标准的 QWidget 实例，并把它作为容器用来“存放”在 calculatorform.ui 中描述的窗体。

```
int main(int argc, char *argv[])
{
    QApplication app(argc, argv);
    QWidget *widget = new QWidget;
    Ui::CalculatorForm ui;
    ui.setupUi(widget);
    widget-&gt;show();
    return app.exec();
}
```

Ui:: CalculatorForm 即是单公有继承自界面实体类的子类，它继承了界面实体类的公有方法 setupUi()，我们调用它实现界面元素的布置。

运行一下这个程序，大家就会发现问题了，它并没有完成加法计算的功能。这种方法的 局限性暴露了出来，由于在 Qt4 中，Qt Designer 只能完成基本的系统内置的信号与槽的连 接功能，而不再提供代码编辑功能，那么我们需要自定义信号和槽时，往往就无能为力了。 这就需要使用单继承法和多继承法来实现与用户的交互，因为它们可以通过代码操纵窗体元 素。

10.3.2 单继承法

本小节例子的代码是来自源码包中的 calculatorform 实例。

1.使用要点

这种做法的要领是，子类化窗体的基类（如 QWidget 或者 QDialog 等），并在该自定义 类中定义一个私有（private）成员变量，该变量是由 uic 工具生成的基于 Qt Designer 生成 的.ui 文件的界面类的实例，其形式为：

```
private: Ui_YourFormName ui
```

或者：

```
private:Ui::YourFormName ui
```

接下来，在该自定义类的构造函数中构造和初始化界面元素（使用 setupUi()函数），这之后，便可以通过实例 ui 来使用界面中的各个部件。

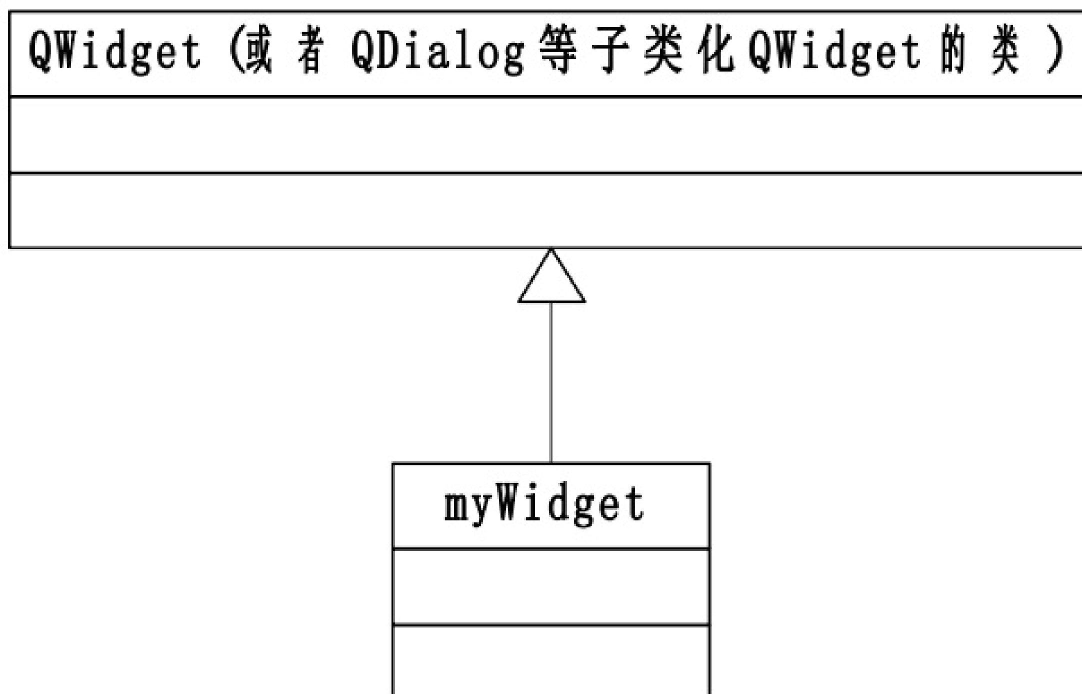


图 10-3 单继承方式的类关系图

2.实例讲解

我们仍然以 Calculator Form 为例，讲解这种方式的用法。首先介绍完全手写代码的方 法，了解了这个后，你可以尝试在 IDE 里面如 Qt Creator 来完成，这就比较容易了。

在引用 Ui::CalculatorForm 之前，我们需要加入由 uic 生成的 C++头文件的声明：

```
#include "ui_calculatorform.h"
```

在工程文件中也要加入 calculatorform.h 这个头文件：

```
HEADERS = calculatorform.h
```

现在看一下我们这个子类的声明：

```
class CalculatorForm : public QWidget
{
    Q_OBJECT
public:
    CalculatorForm(QWidget *parent = 0);
private slots:
    void on_inputSpinBox1_valueChanged(int value);
    void on_inputSpinBox2_valueChanged(int value);
private:
    Ui::CalculatorForm ui;
};
```

第 1 行声明了 CalculatorForm 这个类公有继承自 QWidget，接下来第 9、10 两行声明了一个私有的 ui 变量，以后就由它来完成对应用程序界面的创建 、布置以及信号与槽的关联。再看看构造函数：

```
CalculatorForm::CalculatorForm(QWidget *parent)
: QWidget(parent)
{
    ui.setupUi(this);
}
```

前面讲过，setupUi()是重要的函数，我们一般在类的构造函数中调用它，由它来完成窗体的界面元素的布局和设置。

接下来，就可以使用 ui 变量来访问界面元素了，请看下面的示例代码：

```
void CalculatorForm::on_inputSpinBox1_valueChanged(int value)
{
    ui.outputWidget-&gt;setText(QString::number(value + ui.inputSpinBox2-&gt;value()));
}
```

这种方式的优点是，应用程序能够控制用户界面的外观和显示方式， 并能够与用户进行交互；此外，还可以使用同样的一个 .ui 文件来生成多个不同的自定义界面类，也就是可以 重用 Qt Designer 绘制的用户界面。尤其是当我们需要由一个已经存在的用户界面上派生出多个类似的界面类时，这种方法非常有效。

10.3.3 多继承法

本小节例子的代码是来自源码包中的 multipleinheritance 实例。

1.使用要点

这种做法的要领是，从使用 Qt Designer 设计的界面实体类（Ui::YourFormName）和 QWidget 或者 QDialog 及其子类中多继承派生一个自定义界面类，其形式为：

```
class myWidget : public QWidget,public Ui_YourFormName
{
    Q_OBJECT
public:
    myWidget( QWidget *widget = 0 );
}
```

或者这样写：

```
class myWidget : public QWidget,public Ui::YourFormName
{
    Q_OBJECT
public:
    myWidget( QWidget *widget = 0 );
}
```

这样，我们就可以在 myWidget 这个子类中直接使用界面类的成员，也可以显式的设置信号与槽的连接。比如你在界面中布置了一个 pushButton 类， 它的 ObjectName 属性为 pushButton。那么，在程序中就可以这样使用：

```
pushButton-&gt;setText(tr(“Add”));
```

大家看看，是不是非常的简便。下面这张 UML 图显示了多继承法中的类关系图。

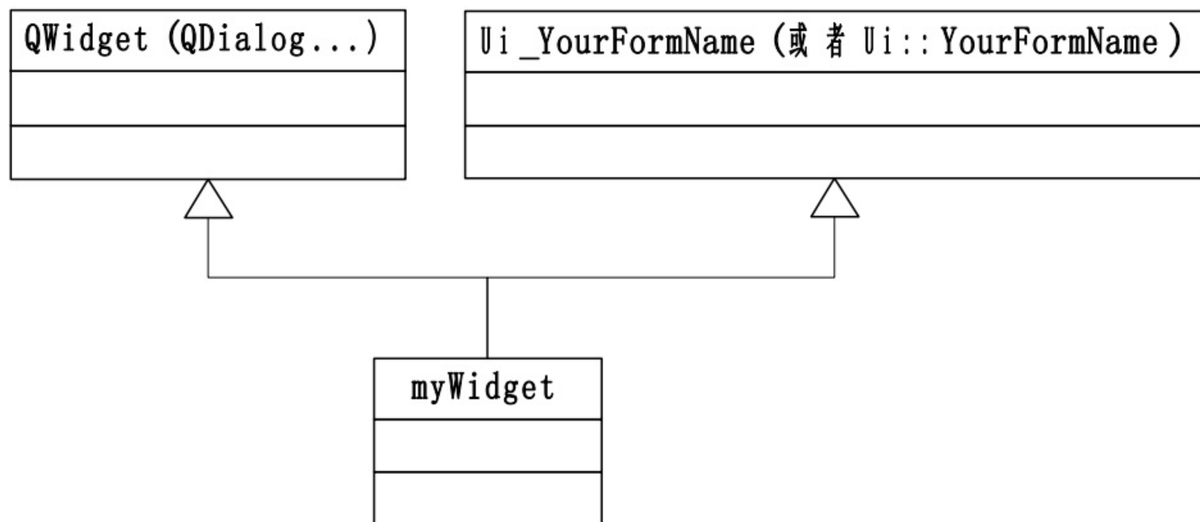


图 10-4 多继承方式-类关系图

2.实例讲解

我们这个实例名叫 Multiple Inheritance ， 完成简单的加法计算功能，.ui 界面文 件与前述相同。

先来看看 calculatorform.h 头文件中的内容：

```
#include "ui_calculatorform.h"
class CalculatorForm : public QWidget, private Ui::CalculatorForm
{
    Q_OBJECT
public:
    CalculatorForm(QWidget *parent = 0);
private slots:
    void on_inputSpinBox1_valueChanged(int value);
    void on_inputSpinBox2_valueChanged(int value);
};
```

首先需要加入由 uic 根据 calculatorform.ui 生成的 ui_calculatorform.h 头文件，因为我们自定义的 CalculatorForm 类的基类之一的 Ui::CalculatorForm 是在该文件中定义的。 在这里我们选择私有继承自 Ui::CalculatorForm 的原因是，以后不准备再以 CalculatorForm 类为基类来创建派生类，因而需要确保界面元素保持私有。当然，你也可以选择公有或者保 护继承，那么就可以再以 CalculatorForm 为基类，继续派生。

接下来与使用单继承方式类似的，我们在构造函数中调用 setupUi()方法完成界面元素 的初始化和布局。

```
CalculatorForm::CalculatorForm(QWidget *parent)
: QWidget(parent)
{
    setupUi(this);
}
```

接下来，我们可以像前几章中介绍的完全用手写代码的方式创建窗体那样直接访问界面元素，而无需 ui 前缀了，请看下面的示例代码：

```
void CalculatorForm::on_inputSpinBox1_valueChanged(int value)
{
    outputWidget-&gt;setText(QString::number(value + inputSpinBox2-&gt;value()));
}
void CalculatorForm::on_inputSpinBox2_valueChanged(int value)
{
    outputWidget-&gt;setText(QString::number(value + inputSpinBox1-&gt;value()));
}
```

主函数可以这样编写：

```
int main(int argc, char *argv[])
{
    QApplication app(argc, argv);
    CalculatorForm calculator;
    calculator.show();
    return app.exec();
}
```

在 main()函数中，我们依次实例化 QApplication 和 CalculatorForm 的对象，然后调用 show()函数来显示窗体。

从以上对 3 种继承方式的分析可见，多继承方式可直接对 ui 页面上的控件或函数进行 操作调用，代码编写更加清晰、简洁；而使用单继承方式，在操作 ui 页面上的控件时需要 加上 ui 对象前缀，编写代码较为繁琐。但对于程序中所需用到的 ui 页面较多时，使用单继 承方式则要简便很多。因此，在实现较为复杂的多窗口程序时，可以优先考虑使用单继承法。 而直接使用法一般采用的比较少。

10.4 运行时加入处理.ui 文件的方法

借助 Qt 提供的 QtUiTools 模块以及其中的 QUiLoader 类我们可以实现在运行时获取并 加载.ui 文件。我们可以使用任何 QIODevice 的实例来读取用户界面架构，通常使用 QFile 类的实例。QUiLoader::load() 方法读取了.ui 文件中包含的界面架构并将它赋予一个 QWidget 子类化的实例。

本节介绍的实例名为 calculatorbuilder，仍然实现简单的加法计算功能，只不过是采 用动态加载.ui 文件的方法完成的。

要实现动态加载 ui，首先需在程序中包含 QtUiTools 模块的头文件：

```
#include &lt;QtUiTools&gt;
```

然后新建一个.qrc 文件，描述.ui 文件的路径：

```
&lt;!DOCTYPE RCC&gt;&lt;RCC version="1.0"&gt;
&lt;qresource prefix="/forms"&gt;
&lt;file&gt;calculatorform.ui&lt;/file&gt;
&lt;/qresource&gt;
&lt;/RCC&gt;
```

将弹出子窗口的.ui 文件放在工程目录下。

接下来，在程序的.pro 文件中手动加入以下两行代码：

```
CONFIG += uitools
RESOURCES += calculatorbuilder.qrc
```

我们对下面的重点代码进行讲解。

```
QUiLoader loader;
QFile file(":/forms/calculatorform.ui");
file.open(QFile::ReadOnly);
QWidget *formWidget = loader.load(&file, this);
file.close();
```

第 1 行新建一个 QUiLoader 实例。

第 2 行指定所需.ui 文件的路径，新建一个 QFile 实例。 第 3 行以只读方式打开此文件。

第 4 行调用 QUiLoader 对象的 load()函数将.ui 文件装载到一个 QWidget 对象中，并将 此 QWidget 对象返回。

第 5 行关闭文件。

大家应该看出来了，采用动态加载.ui 文件的方式的最大好处是我们随时可以调整用户 界面的布局和显示，而不用重新编译应用程序。

10.5 信号与槽的自动连接

信号与槽可以通过使用手写代码显式的实现关联， 也可以运用 QMetaObject 类规定的槽 函数命名范式来实现自动关联。

10.5.1 显式关联

首先我们来看一下，不使用“自动关联规则”的情形。

在下面这段代码里面，我们定义了一个对话框类，它有一个私有的槽 checkValues()， 它用来检验用户提供的值是否正确。

```
class ImageDialog : public QDialog, private Ui::ImageDialog
{
    Q_OBJECT
public:
    ImageDialog(QWidget *parent = 0);
private slots:
    void checkValues();
};
```

在这个对话框类的构造函数中，我们把 Cancel 按钮的 clicked()信号与对话框类的 reject()槽关联起来，把 OK 按钮的 clicked()信号与对话框类的 checkValues()槽关联起来， 这都是通过手写代码显式的实现的。

```
ImageDialog::ImageDialog(QWidget *parent)
: QDialog(parent)
{
    setupUi(this);
    okButton-&gt;setAutoDefault(false);
    cancelButton-&gt;setAutoDefault(false);
    ...
    connect(okButton, SIGNAL(clicked()), this, SLOT(checkValues()));
}
void ImageDialog::checkValues()
{
    if (nameLineEdit-&gt;text().isEmpty())
    {
        (void) QMessageBox::information(this, tr("No Image Name"),
        tr("Please supply a name for the image."), QMessageBox::Cancel);
    }
    else
    {
        accept();
    }
}
```

10.5.2 自动关联

下面的例子演示了信号与槽自动关联的具体实现过程 。我们只需按照下面的标准格式定义槽函数，这之后，uic 将会根据 QMetaObject 类制定的规则，生成界面实体类的 setupUi() 函数的内容，并完成信号与槽的关联，这一过程是隐藏在背后实现的，我们也无需过多关心。

```
void on_&lt;object name&gt;_&lt;signal name&gt;(&lt;signal parameters&gt;);
```

我们通过定义私有槽函数 on_okButton_clicked()即实现了 OK 按钮的 clicked()信号和槽的连接。

```
class ImageDialog : public QDialog, private Ui::ImageDialog
{
    Q_OBJECT
public:
    ImageDialog(QWidget *parent = 0);
private slots:
    void on_okButton_clicked();
};
```

信号与槽的自动连接机制提供了一种槽函数的命名范式，它简化了程序设计者的工作 ，使得用户界面设计变得有章可循并且充满乐趣。笔者建议读者朋友尽量采用这种方法来设计 Qt 用户界面。

10.6 问题与解答

问：动态加载方式与编译时加载.ui 文件的方式，在原理上有什么不同？ 答：从原理上来讲，动态加载方式并不需要通过 uic 把.ui 文件转换成 C++代码，它是程序运行时的时候使用 QUiLoader 类载入该文件的，而编译时加载.ui 文件的方法，是需要借助 uic 把.ui 文件中的内容转化为 C++代码的，并生成 ui_xxx.h 文件。就像下面这种方式：

问：使用动态加载方式时，我如何访问窗体中的各个子窗口部件呢？ 答：可以通过调用全局函数 qFindChild<T>()来访问这个窗体中的各个子窗口部件，举例如下：

```
ui_findButton = qFindChild&lt;QPushButton*&gt;(this, "findButton");
ui_textEdit = qFindChild&lt;QTextEdit*&gt;(this, "textEdit");
ui_lineEdit = qFindChild&lt;QLineEdit*&gt;(this, "lineEdit");
```

在我们提供的 calculatorbuilder 例子里面有这个函数的具体用法，大家可以参考，更为详细的内容请查看 Qt Assistant。

也可以使用 QObject::findChild<T>来访问这个窗体中的各个子窗口部件，举例如下：

```
QButton *button = myWidget-&gt;findChild&lt;
QButton *&gt;( tr(“ok”) );
if (button)
{
    ...
}
```

这里的 findChild<T>()函数是一个模板成员函数，它可以返回与给定的名字和类型相匹配的子对象。注意，由于受编译器的制约，它不能在 MS VC6 中使用。

10.7 总结与提高 本章主要讲解了以下内容：

- uic 的使用方法
- ui_yourformname.h 文件的组成和功用
- 在编译时加入处理.ui 文件的方法，包括直接使用法、单继承法和多继承法
- 在运行时加入处理.ui 文件的方法，主要是动态加载的方法
- 这些方法的优缺点以及最佳使用场合
- 掌握信号与槽实现自动关联的方法

这其中的关键是理解 ui_xxx.h 文件的组成和功用，以及怎样根据应用程序的特点灵活 的选择最适合的集成.ui 文件的方法。根据笔者的体会，当应用程序的规模较大，并且界面 繁多时，使用单继承法比较合适；而应用程序界面比较少时，使用多继承法则较好，或者使 用动态加载.ui 文件的方法也可；而直接使用.ui 文件的情形比较少见。建议读者朋友把本章 中举的例子都逐一实验一下，以达到熟能生巧的目的。

第 11 章 布局管理

本章重点

- 在窗体中摆放窗口部件的方法
- 布局的基本概念和分类
- 基本布局的创建方法和步骤
- 堆栈窗体的创建和使用
- 切分窗口的创建和使用
- 各种布局方式的综合使用
- 如何优化布局结构
- 布局管理的经验总结

放置在窗体中的每一个窗口部件都必须有一个合适的大小和位置，并且它们应该能够随着程序自身的辩护做出响应而不改变整体的布局结构。 Qt4 提供了多种用于在窗体中摆放窗口部件的类：QHBoxLayout、QVBoxLayout、QGridLayout、QFormLayout 和 QStackedLayout。这些类简单易用，几何每个 Qt 开发人员都会用到它们-或者直接在源代码中，或者通过 Qt Designer。在这一章里，我们将学习如何使用 Qt4 中基本的布局管理器以及栈部件和分裂器部件来完成一些比较复杂的布局。

可以执行布局管理功能的其他类还有 QSplitter、QScrollArea、QMainWindow 和 QMdiArea 等。这些类所拥有的共同点在于它们提供了一种用户可以灵活掌控的布局方式。 例如，QSplitter 就提供了一个切分窗口拖动条（splitter bar），通过拖拽它，用户可以 改变窗口部件的大小。QMdiArea 则为多文档界面（multiple document interface,MDI）提 供了支持。它们经常适合用作布局类的替换方式，我们将在本章中讲述其中的部分内容。

11.1 基本概念和方法

Qt 布局管理器是 Qt 图形用户界面设计的基本内容，在前面的几章中也断断续续的讲到 了 Qt 的布局管理器的使用，也做了一些简单的应用。本小节将详细介绍 Qt 的布局管理器，详细阐述 Qt 布局管理器的一些特性。

11.1.1 摆放窗口部件的方法

Qt 提供了几种在窗口部件上管理子窗口部件的基本方式。

一共有 3 种方法用于管理窗体上子窗口部件的布局：绝对位置法、人工布局法和布局管理器法。

相比于使用固定尺寸和位置，布局提供了功能强大且极具灵活性的另一种方案。使用 布局后，编程人员无需计算尺寸和位置，布局可以自动进行调整，符合用户屏幕、语言以及 字体的要求。

1.绝对位置法

这种方法是最原始的摆放窗口部件的方法，甚至都不能称其为 “摆放”。它对窗体的 各个子窗口部件分配固定的大小和位置，是通过调用基类 QWidget 提供的 setGeometry()函 数来实现的。

绝对位置法有很多缺点：

- 用户无法改变窗口的大小，当父窗口改变时，子窗口不能做出相应的调整。
- 如果用户选择了一种不大常用的大字体，或者当应用程序被翻译成另外一种语言 时，也许会把一些文本截断。
- 对于某些风格的平台，这些窗口部件可能会具有并不合适的尺寸大小。
- 必须人工计算这些位置和大小。这样做不仅非常枯燥而且极易出错，并且还会让后 期的维护工作变得痛苦万分。

很显然，使用这种方式管理 GUI 应用程序大大降低了程序员的开发效率，降低了应用 程序的质量和适应性。

2.人工布局法

这种方法的核心是通过重载 QWidget::resizeEvent(QResizeEvent*)函数来使得子窗口 的的大小尺寸总是和父窗口的大小成比例，也就在一定程度上减轻了计算量，但是在其中也 要通过 setGeometry()函数来设置子窗口部件的位置和大小。

在程序的规模比较小，并且不需要时常变更设计的情况下，绝对位置法勉强可以胜 任。但是它就像前面的绝对位置法一样，仍然需要计算许多手写代码中的常量，尤其是当设 计被改变的时候，这种情况更加突出，而且它并没有消除文本会被截断的危险。辅以社会自 子窗口部

件的大小提示，应该可以规避这种风险，但是这样会使代码变得尤为复杂。

3.布局管理器法

这种方式是使用 Qt 设计用户界面、组织管理 Qt 窗口部件的最好方法。布局管理器为 窗口部件提供了有感知的默认值（sensible default sizes），可以随着窗口部件大小的变 化，对子窗口部件的大小和位置做出适当的调整。

11.1.2布局管理器

Qt 的布局管理器负责在父才窗口部件区域内构建子窗口部件。这些管理器可以使其中 的窗口部件自动定位并重新调整子窗口部件、保持窗口部件敏感度最小化的变化和默认尺 寸，并可在内容或文本字体更改时自动重新定位。在 Qt Designer 中，完全可以使用布局管理器来定位控件。图 1-1 显示了使用了顶层布局管理器的应用程序在不同尺寸下的情形，可以看到，由于布局管理器的存在，使得窗口部件保持了整体的布局。



图 11-1a 图 11-1b

图 11-1 以不同尺寸显示的同一对话框

1.布局类的继承关系

QLayout 类是 Qt 的几何管理器的基类，它派生自 QObject 类和 QLayoutItem 类，是一 个抽象基类，必须被派生类所重新实现。它的派生类主要有 QBoxLayout， QGridLayout, QFormLayout 以及 QStackedLayout。而 QBoxLayout

又有两个主要的 Qt4 子类，QHBoxLayout 和 QVBoxLayout。它们之间的继承关系如图 11-2 所示，图中的抽象类用斜体表示。

- QObject
    - QLayout
        - QGridLayout
        - QHBoxLayout

图 11-2 Qt4 布局管理器类继承关系图

QLayout 也提供了一些有用的方法，如 setSizeConstraint() 和 setMenuBar()等,但是我们很少会用到它们。

2.内建布局管理器

Qt 的内建布局管理器可以将窗口部件以及其他布局排列为横向，纵向，网格以及表单中。

如前所述，在截至目前的 Qt 4.5 中，主要提供了 5 种内建的布局管理器。

(1) 水平布局管理器（QHBoxLayout） 按从左至右的顺序将管理的窗口部件横放在一行中。

图 11-3 显示了使用水平布局管理器的窗口的大致情形。

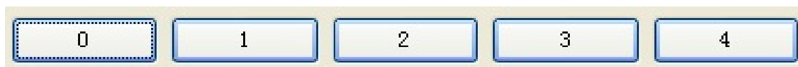


图 11-3 使用 QHBoxLayout 布局管理器排列的窗口部件

(2) 垂直布局管理器（QVBoxLayout） 按从上至下的顺序将管理的窗口部件竖放在一列中。

图 11-4 显示了使用水平布局管理器的窗口的大致情形。



图 11-4 使用 QVBoxLayout 布局管理器排列的窗口部件

(3) 栅格布局管理器（QGridLayout） 把管理的窗口部件放在可扩展的单元格中，这样一来，如有必要， 窗口部件可以跨越多个单元。

图 11-5 显示了使用水平布局管理器的窗口的大致情形。

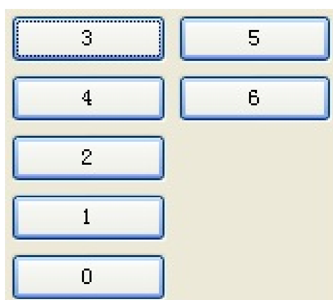


图 11-5 使用 QGridLayout 布局管理器排列的窗口部件

(4) 表单布局管理器（QFormLayout） 表单布局管理器主要用作管理界面上的输入窗口部件（input widgets）以及和它们相连的标签窗口部件（labels）。

图 11-6 显示了使用水平布局管理器的窗口的大致情形。

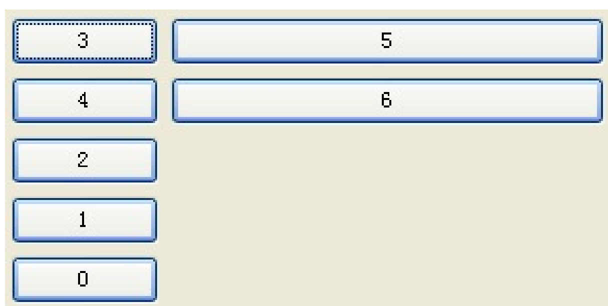


图 11-6 使用 QFromLayout 布局管理器排列的窗口部件

(5) 栈布局管理器（QStackedLayout） 按照一种类似于栈的方式排列窗口部件，在某一时刻只有一个窗口部件是可见的。 图 11-7 显示了使用水平布局管理器的窗口的大致情形。

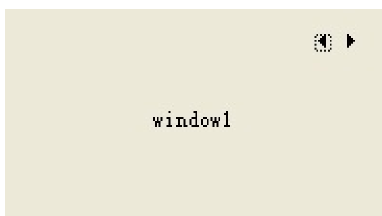


图 11-7 使用 QStackedLayout 布局管理器排列的窗口部件

小贴士：截至目前的 4.5.2 版，Qt Designer 的窗口部件盒没有可视化的提供对栈布局管理 器的支持，不过它提供了一个栈部件 QStackedWidget，作用与栈布局管理器类似。因此， 在使用 Qt Designer 绘制 GUI 界面时，完全可以使用 QStackedWidget 来代替 QStackedLayout。

每个内建布局管理器均支持窗口部件在分配的范围内横向/纵向对齐，这样只需使用简单的布局和对齐属性，即可自定义用户界面的外观。

除了上述内建的布局管理器外，分裂器也是一种常见的布局管理器，它是由 QSplitter 类实现的。在 Qt Designer 的窗口部件盒中也没有为它提供可视化的部件，不过却提供了 鼠标右键上下文菜单项来实现该布局。在 Qt 4.5 的后续版本中，它有望“转正”，成为正 式的内建布局管理器。在本章后面的内容中会详细的讲到如何使用分裂器布局。分裂器布局 又分为分裂器水平布局和分裂器垂直布局，图 11-8 和图 11-9 分别显示了它们的大致情 形。

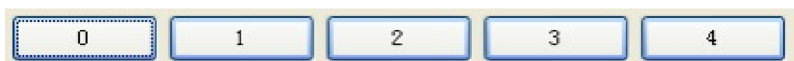


图 11-8 使用分裂器水平布局管理器排列的窗口部件



图 11-9 使用分裂器垂直布局管理器排列的窗口部件

在实际的应用程序开发中，很少有只用到单一布局的情况，多数情况下需要组合或者 嵌套应用布局。Qt4 中的各种布局可以被组合或嵌套至任意的级别中，在后面的例子中将为 大家展示如何将基本布局综合使用起来。

11.1.3优化布局结构

在迄今为止讲到每一个例子中，我们只是简单的把窗口部件放置到某个确定的布局中。但在某些情况下，由此形成的布局看起来可能还不是我们最想要的形式。在这些情形中，可以通过改变要摆放的窗口部件的大小策略和大小提示来调整布局。

1.大小提示（size hint）和最小大小提示（minimum size hint）

在介绍 Qt 窗口部件的大小策略之前，首先介绍大小提示（ size hint）和最小大小提 示（minimum size hint）。

(1) 大小提示

大小提示是 Qt 为一个窗口部件推荐的尺寸。当 Qt GUI 窗口部件进行初始化时，将通 过 QWidget::sizeHint()来获得窗口部件的大小提示，这是一个虚函数，它的原型为：

```
virtual QSize sizeHint () const
```

在未被重载的情况下，它的返回值是这样的：

- 如果该窗口部件不属于任何布局管理器，那么该函数将返回一个无效的值；
- 如果该窗口部件属于某个布局管理器，那么该函数将返回一个该布局管理器认为比 较合适的尺寸。

(2) 最小大小提示

最小大小提示（minimum size hint）是 Qt 为窗口部件推荐的最小尺寸，它的使用规 则是：

如果需要绘制的窗口部件的尺寸（包括长和高两个方面）小于其最小提示（这在 Qt Designer 中往往表现为有些被压缩的看不到它的内容），并且该窗口部件的最小提示在最 大尺寸和最小尺寸允许的范围内，那么该窗口部件显示的尺寸将是其最小提示的值。

设置窗口部件的最小大小提示是通过 QWidget::minimumSizeHint()完成的。它的返回 值有如下情景：

- 如果该窗口部件没有布局管理器，该函数返回一个无效的值；
- 如果该窗口部件属于某个布局管理器，该函数返回布局管理器认为合适的一个尺 寸。

2.大小策略（size policy）

一个窗口部件的大小策略会告诉布局系统应该如何对它进行拉伸或收缩。 Qt 为它所有的内置窗口部件都提供了合理的默认大小策略值，但是由于不可能为每一种可能产生的布局都提供唯一的默认值，所以在一个窗体中，开发人员改变它上面的一个或两个窗口部件的大 小策略是非常普遍的现象。一个 QSizePolicy 既包含一个水平分量也包含一个垂直分量。 以下是一些常用的取值：

表 11-1 枚举值 QSizePolicy::Policy 的内容

枚举常量	值	说明
QSizePolicy::Fixed	0	大小提示是该窗口部件的唯一尺寸选择，所以它不会 发生任何的伸缩。
QSizePolicy::Minimum	GrowFlag	大小提示是该窗口部件的最小尺寸，它不会变得更 小，但它可以变得更大，不过采用该策略的窗口部件 在“争夺”空间上不占优势。
QSizePolicy::Maximum	ShrinkFlag	大小提示是该窗口部件的最大尺寸，也就是该窗口部 件不会比大小提示的尺寸更大。该窗口部件可以在没 有受到其它窗口部件“要求”的情况下，自由的缩小 尺寸。
QSizePolicy::Preferred	GrowFlag | ShrinkFlag	一般情况下，该窗口部件会将大小提示作为它的优先 和最佳选择，但它也可以变得足够的小，也可以变大，但不占优势。该策略是 QWidget 窗口部件默认的 策略。
QSizePolicy::Expanding	GrowFlag | ShrinkFlag | ExpandFlag	采用该策略的窗口部件也能够感觉到尺寸提示，但是 它倾向于尽可能的占用更大的空间，该窗口部件也可 以变得足够小。
QSizePolicy::MinimumExpanding	GrowFlag | ExpandFlag	大小提示将是该窗口部件的最小尺寸，该窗口部件将 尽可能的占用更多的空间。该策略已经不再被推荐使 用，建议用 Expanding 替代它，并且重载 minimumSizeHint()。
QSizePolicy::Ignored	ShrinkFlag | GrowFlag | IgnoreFlag	与 Expanding 有些相似，只是所有的大小提示都被忽 略，该窗口部件将会尽可能的占用空间。

表 11-1 中的“值”这一列实际上告诉了我们每一种策略一般是具有 “倾向性”的，比

如 QSizePolicy::Fixed 的值为 0，则它“倾向于”保持自己的大小不变，即保持大小提示 的尺寸。而 QSizePolicy::Expanding 的值是 3 个值的叠加，总的“倾向性”是趋于占用更 多空间的，等等。这就为当多个具有不同大小策略的窗口部件放置在一起时，如何判断它们 占用空间的模式提供了基本的判断依据，以下是几种常见的组合。

相同大小策略的窗口部件被布局管理器组合在一起。在这种情况下，除了窗口部件 不能超出它的大小范围外，不同的窗口部件可以按自己的伸缩因子在其允许的范围内自由的 伸缩。

QSizePolicy::Fixed 和任何其他的大小策略组合在一起。

具有 QSizePolicy::Fixed 大小策略的窗口部件其大小是不变的，即保持在 sizeHint() 大小，而其他的窗口部件可以在允许的范围内自由伸缩。

QSizePolicy::Preferred 和 QSizePolicy::Expanding 组合在一起。

具有 QSizePolicy::Preferred 尺寸策略的窗口部件其大小是不变的，即它认为大小提 示是最适合它的，而其他的窗口部件大小可以在其允许的范围内自由伸缩。

QSizePolicy::Ignored 和其他尺寸策略（QSizePolicy::Fixed 策略除外）组合在 一起的时候，不同的窗口部件在各自允许的范围内自由伸缩。

QSizePolicy::Preferred，QSizePolicy::Minimum 和 QSizePolicy::Maximum 组合 在一起的时候，各窗口部件在各自允许的范围内可以自由伸缩。

3.伸缩因子（stretch factor）

除了大小策略中包含的水平方向和垂直方向两个分量之外， QSizePolicy 类还保存了水平方向和垂直方向的一个伸缩因子。这些伸缩因子可以用来说明在增大窗体时，对不同的子窗口部件应使用的不同放大比例。即需要设置 QSizePolicy::horizontalStretch 和 QSizePolicy::verticalStretch 的值来实现。默认情况下，被布局管理器组合在一起的窗口部件的伸缩因子是相等的，都为 0。此时，在所有的窗口部件都没有超出各自的大小范围 允许的情况下，窗口部件的大小始终相等。

例如，假定在一个 QListWidget 的右面还有一个 QTextEdit，并且希望这个 QTextEdit 的长度能够是 QListWidget 长度的两倍，那么就可以把这个 QTextEdit 在水平方向上的拉 伸因子（QSizePolicy::horizontalStretch）设置为 2，而把 QListWidget 在水平方向上的 拉伸因子（QSizePolicy::horizontalStretch）设置为 1；垂直方向上保持默认为 0，即两 者一样的高。这样设置的效果如图 11-10 所示。

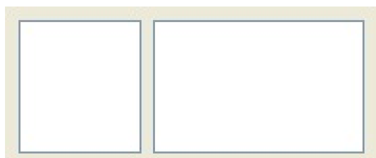


图 11-10 设置伸缩因子后窗体的效果

4.大小约束（size constraint）

影响布局方式的另一种方法是设置它的子窗口部件的最大大小、最小大小或固定大小。这些是通过设置 sizeConstraint 属性来完成的。该属性值是一个枚举常量，定义了布局的大小约束的模式。表列出了它所有可能的取值，它的默认值是 QLayout::SetDefaultConstraint。获取和设置该属性值可以通过 QWidget::layout()来获取主窗口部件的布局管理器，然后可以调用

QLayout::sizeConstraint()函数来查看当前的设置情况，然后再通过 QLayout::setSizeConstraint()函数来设置该布局管理器的 sizeConstraint 属性。这两种函数的原型如下：

```
SizeConstraint sizeConstraint () const void setSizeConstraint ( SizeConstraint )
```

其中，SizeConstraint 的取值即是在表 11-2 中的枚举值的范围内。

表 11-2 布局管理器的大小约束属性（QLayout::SizeConstraint）可能的取值

常量	值	说明
QLayout::SetDefaultConstraint	0	主窗口部件的最小尺寸设置为 minimumSize(), 除非该窗口部件已经有一个 最小尺寸
QLayout::SetFixedSize	3	主窗口部件的尺寸设置为 sizeHint()，并且不允许改变该窗口部件的尺寸
QLayout::SetMinimumSize	2	主窗口部件的最小尺寸设置为 minimumSize(), 并且该窗口部件不能够变得 更小
QLayout::SetMaximumSize	4	主窗口部件的最大尺寸设置为 maximumSize(), 并且该窗口部件不能够变得 更大
QLayout::SetMinAndMaxSize	5	主窗口部件的最小尺寸设置为 minimumSize(), 最大尺寸设置为 maximumSize()
QLayout::SetNoConstraint	1	主窗口部件的大小不会受到约束

5.空白（margin）和间距（spacing）

每种布局都有两个重要的属性，空白和间距。空白指的是整个布局四周距离窗体边缘的距离；间距指的是布局管理器内部各个窗口部件之间的距离。

空白属性即 margin()，间距属性即 spacing()，它们的默认值是有窗体的风格决定 的。Qt 的默认风格下，子窗体部件的 margin()的值是 9 英寸，窗体的 margin()值是 11 英 寸。spacing()的值与 margin()相同。

如果要设置这两个值可以通过 setMargin()和 setSpacing()。

注意，从 Qt4.3 开始，margin()属性已经逐渐不再被 Qt4 所推荐，更好的设置空白的 方法是使用 setContentsMargins()方法，它的原型如下：

```
void QLayout::setContentsMargins ( int left, int top, int right, int bottom )
```

其中，left, top, right, 和 bottom 表示环绕在布局周围的空白。

对于 QGridLayout 和 QFormLayout，不要使用 setSpacing()方法，而是要分别使用 setHorizontalSpacing()和 setVerticalSpacing()方法来设置水平和垂直方向的间距。如 果你使用了 setSpacing()方法，获取 spacing()时，它的返回值将为-1。

11.2在 Qt Designer 中使用布局

为了确保界面元素在应用程序程序在运行时或被预览时的各种状态下都能够正常显示，我们需要把它们放进布局当中去。

11.2.1 应用和破除布局

应用布局的最简单做法是选中界面元素，使用工具栏上的按钮、鼠标右键的上下文菜单，以及【Form】菜单都可以实现。

一旦界面元素被放进一个布局之中，它就不能单独自由行动了 -你不可以单独改变它的 大小，因为布局接管了这一工作，它控制了位于其中的界面元素的几何以及间隔器的大小策 略提示（size hint）。所以你要么破除布局，人工调整界面元素的大小，要么通过调整布 局的大小来间接调整界面元素的大小。

要破除一个布局，可以使用快捷键 Ctrl+0 或者通过鼠标右键上下文菜单以及主菜单项 或工具栏按钮。在一个布局完成之后，你仍然可以向其中添加或删除间隔器来影响布局内部 的窗口部件的几何，最为便捷的方式是从部件选择器中拖出一个间隔器，并把它拖入到布局 之中，删除的时候则相反，把间隔器从布局中拖出来即可。

1. 向布局中增加窗口部件

如图 11-11 所示，要向一个已经存在的布局中增加窗口部件，只需要选中并拖动该窗口部件，把它从当前位置拖进布局之中，并在认为合适的位置放开鼠标。注意，当你的拖动 窗口部件在布局上方停留时，布局内部会显示一条蓝色的线，它提示了你的窗口部件未来在布局中的位置。

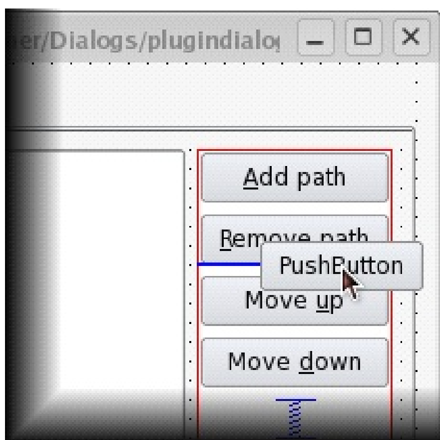


图 11-11 向布局中增加窗口部件

2. 设置一个顶级布局（Top Level Layout） 设置界面的顶级布局是很有必要的，它能确保窗口界面元素在应用程序的各种状态下均能够保持适当的大小。

要设置顶级布局，需要用鼠标左键选中该 Form，然后使用快捷键或工具栏或者主菜单 项来选中一种布局。

如何验证已经设置了顶级布局呢，简单的做法就是在 Qt Designer 的预览窗口（按下 Ctrl+R 键）中，使用鼠标左键拖动窗口的边缘手柄，查看界面元素的变化情况。如果一切 正常，那就表示你已经设置过了。还有一种做法，就是用鼠标左键点击界面表单（你的 Form），然后在对象查看器中，如果看到 Form 前面有常见的几种布局图标之一（图 11-12 中是垂直布局），那么就表示你已经设置了顶级布局，如图 11-12 所示；反之，如果 Form 前面的图标右下角带有一个红色的停止标志，就表示还没有设置顶级布局，如图 11-13 所 示。

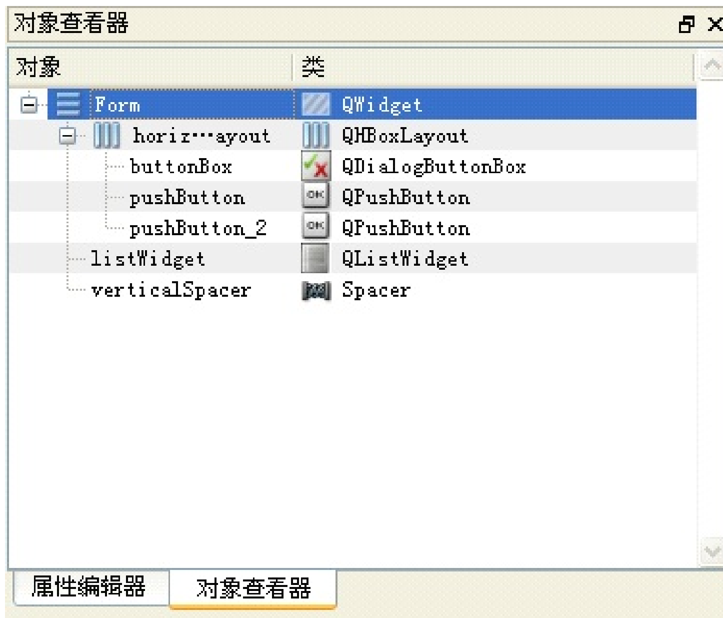


图 11-12 在对象查看器中浏览顶级布局-存在顶级布局

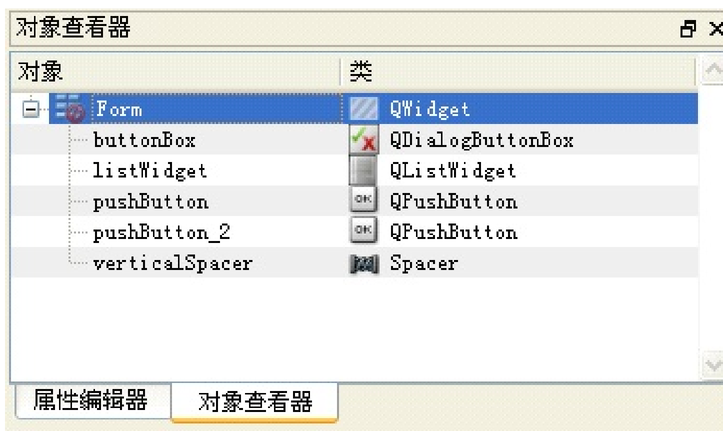


图 11-13 在对象查看器中浏览顶级布局-不存在顶级布局

3. 应用一个布局

要应用一个布局，你可以使用工具栏上的按钮，如图 11-14 所示，也可以使用鼠标右键的上下文菜单，如图 11-15 所示。



图 11-14 使用工具栏按钮应用一个布局

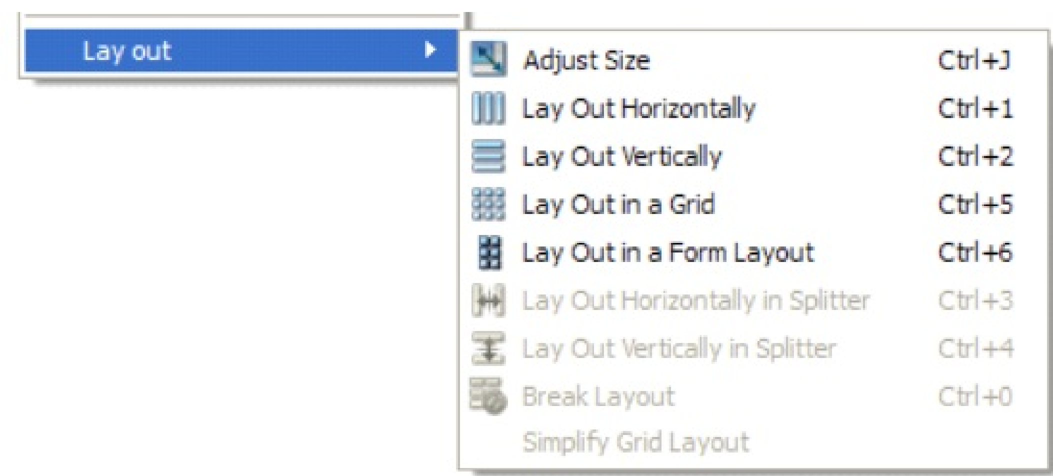


图 11-15 使用鼠标右键的上下文菜单来应用一个布局

11.2.2 快捷键

除了使用标准工具栏以及上下文菜单，我们还可以使用快捷键来对布局进行操作。表 11-3 显示了常见的布局操作所对应的快捷键。

表 11-3 布局的快捷键

布局	快捷键	说明
水平布局	Ctrl+1	将选中的界面元素置于一个水平布局中
垂直布局	Ctrl+2	将选中的界面元素置于一个垂直布局中
栅格布局	Ctrl+5	将选中的界面元素置于一个栅格布局中
表单布局	Ctrl+6	将选中的界面元素置于一个表单布局中
分裂器水平布局	Ctrl+3	创建一个分裂器水平布局，并将选中的界面元素置于其中
分裂器垂直布局	Ctrl+4	创建一个分裂器垂直布局，并将选中的界面元素置于其中
调整大小	Ctrl+J	调整布局的大小，以使得位于其中的元素能够恰当的显示自身内容。关于这方面的内容，可以参见 QWidget::adjustSize()函数
破除布局	Ctrl+0	破除选中的布局

11.3 基本布局实践

基本布局主要包括：水平布局、垂直布局、栅格布局和表单布局这 4 种。在讲解水平 布局和垂直布局之前，还要重点说一下它们的父类 -QBoxLayout，它也是使用比较多的，并 且包含了水平布局和垂直布局的一些共性特点。

11.3.1 QBoxLayout

使用 QBoxLayout 类可以创建一个布局，能够把其中的窗口部件水平的或者垂直的按照直线排列。

之所以取名为“Box”，是由于 QBoxLayout 类把位于其内的空间均匀的划分成若干个 “盒子”，并把各个窗口部件都放入一个盒子里面。

使用 QBoxLayout 创建的布局是有方向性的，主要是有水平和垂直两个方向。当给定其 方向参数为 Qt::Horizontal 时，即表示水平方向，位于其中的窗口部件将水平排列成一 行，并且它们都会找到适合自己的大小。当给定方向参数为 Qt::Vertical 时，窗口部件将按 照垂直直线排列。布局内的其余空间是共享的，它们的尺寸可以由伸缩因子（ stretch factors）来确定。

使用 QBoxLayout 类来创建一个布局是很容易的，做法是直接调用其构造函数，其原型 如下：

```
QBoxLayout::QBoxLayout ( Direction dir, QWidget * parent = 0 )
```

该构造函数它有两个参数，一个是布局的方向，一个是父窗口指针，默认为 0 即当前窗口。布局的方向枚举值如表 11-4 所示。

表 11-4 QBoxLayout 的方向枚举值

常量	值	说明
QBoxLayout::LeftToRight	0	从左到右水平排列
QBoxLayout::RightToLeft	1	从右到左水平排列
QBoxLayout::TopToBottom	2	从上到下垂直排列
QBoxLayout::BottomToTop	3	从下到上垂直排列

一个实例代码如下：

```
//第 1 步
QWidget *window = new QWidget;
QPushButton *button1 = new QPushButton(tr("One"));
QPushButton *button2 = new QPushButton(tr("Two"));
QPushButton *button3 = new QPushButton(tr("Three"));
QPushButton *button4 = new QPushButton(tr("Four"));
QPushButton *button5 = new QPushButton(tr("Five"));
//第 2 步
QBoxLayout *layout = new QBoxLayout(QBoxLayout::LeftToRight,0);
//第 3 步
layout-&gt;addWidget(button1);
layout-&gt;addWidget(button2);
layout-&gt;addWidget(button3);
layout-&gt;addWidget(button4);
layout-&gt;addWidget(button5);
//第 4 步
window-&gt;setLayout(layout);
//显示窗口
window-&gt;show();
```

创建这种布局的一般步骤是：

第 1 步，创建要使用的窗口部件。

第 2 步，创建布局，并指定其方向和父窗口。

第 3 步，将窗口部件依次加入到布局中。

第 4 步，为应用程序程序窗口设置布局。

如上代码所示，在其中通过注释指出了各步骤对应的代码行。这段代码将创建 1 个名 为 window 的应用程序窗口，1 个名为 layout 的 QBoxLayout 实例，以及 5 个按钮，布局方 向是从左到右的水平方向。图 11-16 显示了这个实例的效果，你会发现它与使用水平布局 的效果是一样的。

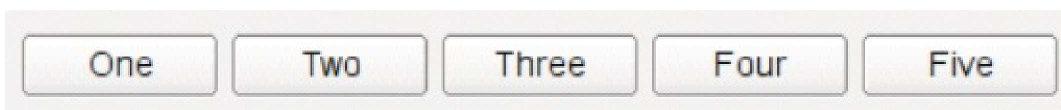


图 11-16 使用 QBoxLayout 类创建的布局效果

当然，在实际应用中，我们大多数情况下会直接使用 QBoxLayout 的两个派生类，水平 布局和垂直布局，它们更加方便而且有针对性。

11.3.2 水平布局

水平布局把在其中的窗口部件按照一条直线水平排列。 QHBoxLayout 类用来创建水平 布局的实例。

一个简单的手写代码创建水平布局的实例如下：

```
QWidget *window = new QWidget;
QPushButton *button1 = new QPushButton(tr("One"));
QPushButton *button2 = new QPushButton(tr("Two"));
QPushButton *button3 = new QPushButton(tr("Three"));
QPushButton *button4 = new QPushButton(tr("Four"));
QPushButton *button5 = new QPushButton(tr("Five"));
QHBoxLayout *layout = new QHBoxLayout;
layout-&gt;addWidget(button1);
layout-&gt;addWidget(button2);
layout-&gt;addWidget(button3);
layout-&gt;addWidget(button4);
layout-&gt;addWidget(button5);
window-&gt;setLayout(layout);
window-&gt;show();
```

使用手写代码创建水平布局大体需要下面这些步骤：

第 1 步，创建布局内的窗口部件。

第 2 步，创建一个水平布局的实例，也就是创建一个 QHBoxLayout 的实例，并将第 1 步创建的窗口部件加入到该布局之中。

第 3 步，调用 QWidget::setLayout()函数将该布局安装到窗体上。设置了布局之后， 在该布局内的窗口部件将以 window 作为它们的父窗口。该实例的效果如图 11-17 所示：

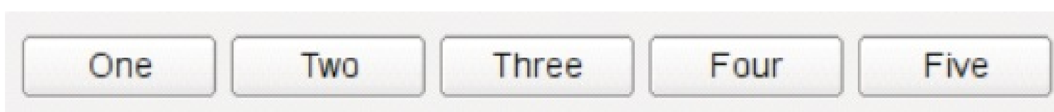


图 11-17 水平布局效果

11.3.3 垂直布局

垂直布局把位于其中的窗口部件按照一条直线垂直排列。 QVBoxLayout 类用来创建一个 垂直布局的实例。

一个简单的使用手写代码创建垂直布局的实例如下：

```
QWidget *window = new QWidget;
QPushButton *button1 = new QPushButton("One");
QPushButton *button2 = new QPushButton("Two");
QPushButton *button3 = new QPushButton("Three");
QPushButton *button4 = new QPushButton("Four");
QPushButton *button5 = new QPushButton("Five");
QVBoxLayout *layout = new QVBoxLayout;
layout-&gt;addWidget(button1);
layout-&gt;addWidget(button2);
layout-&gt;addWidget(button3);
layout-&gt;addWidget(button4);
layout-&gt;addWidget(button5);
window-&gt;setLayout(layout);
window-&gt;show();
```

使用手写代码创建垂直布局大体需要下面这些步骤：

第 1 步，创建布局内的窗口部件

第 2 步，创建一个垂直布局的实例，也就是创建一个 QVBoxLayout 类的实例，并将第 1 步创建的窗口部件加入到该布局之中

第 3 步，调用 QWidget::setLayout()函数将该布局安装到窗体上。设置了布局之后， 在该布局内的窗口部件将以 window 作为它们的父窗口。该实例的效果如图 11-18 所示：

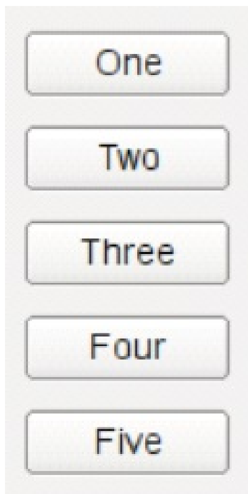


图 11-18 垂直布局实例效果

水平和垂直布局可以嵌套、组合使用，并且可至任意的深度。但在比较复杂的情况 下，使用栅格布局往往是更理想的做法。

11.3.4 栅格布局

栅格布局将位于其中的窗口部件放入一个网状的栅格之中。栅格布局是这样工作的：

它计算了位于其中的空间，然后将它们合理的划分成若干个行（ row）和列（column），并把每个由它管理的窗口部件放置在合适的单元之中，这里所指的单元（ cell）即是指由行和列交叉所划分出来的空间。

在使用栅格布局之前，需要在代码中包含如下的头文件声明：

```
#include &lt;QGridLayout&gt;
```

创建栅格布局的大致步骤如下：

第 1 步，创建布局内的窗口部件。

第 2 步，创建栅格布局的实例，也就是创建一个 QGridLayout 类的实例，并将第 1 步 创建的窗口部件加入到布局之中。

第 3 步，调用 QWidget::setLayout()函数将布局安装到窗体上。 一个典型的实例代码如下：

```
nameLabel = new QLabel(tr("&Name:"));
nameLabel-&gt;setBuddy(nameLineEdit);
ageLabel = new QLabel(tr("&Age:"));
ageLabel-&gt;setBuddy(ageSpinBox);
QGridLayout *gridLayout = new QGridLayout;
gridLayout-&gt;addWidget(nameLabel, 0, 0);
gridLayout-&gt;addWidget(nameLineEdit, 0, 1);
gridLayout-&gt;addWidget(ageLabel, 1, 0);
gridLayout-&gt;addWidget(ageSpinBox, 1, 1);
setLayout(gridLayout);
```

我们来讲解一下这段代码。可以明显的看出， QHBoxLayout 和 QVBoxLayout 的用法相当简单明了，但是 QGridLayout 的用法则稍微有些麻烦。QGridLayout 的工作基于一个二维单 元格。在这个布局中，左上角的 QLabel 即 nameLabel 的位置是(0,0)，而与之相应的 QLineEdit 即 nameLineEdit 的位置是(0,1)。以此类推，ageLabel 的位置是(1,0)，而 ageSpinBox 的位置是(1,1)。它们都占用一列的值。

对于 QGridLayout::addWidget()的调用遵循如下的语法形式：

```
layout-&gt;addWidget(widget,row,column,rowSpan,columnSpan);
```

其中,widget 是要插入到布局中的子窗口部件，(row,column)是由该窗口部件所占用的左上角单元格，rowSpan 是该窗口部件要占用的行数，而 column 是该窗口部件要占用的列 数。如果省略了这些参数，则它们将会取默认值 1。

该示例的运行效果如图 11-19 所示。

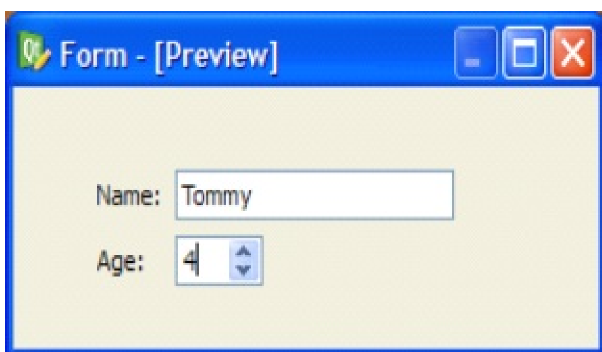


图 11-19 栅格布局效果

在栅格布局中，行和列本质上是相同的，只是叫法不同而已。下面我们将重点讨论 列，这些内容当然也适用于行。

在栅格布局中，每个列（以及行）都有一个最小宽度（ minimum width）以及一个伸缩 因子（stretch factor）。最小宽度指的是位于该列中的窗口部件的最小的宽度，而伸缩因 子决定了该列内的窗口部件能够获得多少空间。它们的值可以通过 setColumnMinimumWidth()和 setColumnStretch()方法来设置。

此外，一般情况下我们都是把某个窗口部件放进栅格布局的一个单元中，但窗口部件 有时也可能会需要占用多个单元。这时就需要用到 addWidget()方法的一个重载版本，它的 原型如下：

```
void QGridLayout::addWidget ( QWidget * widget, int fromRow, int fromColumn, int rowSpan,
```

这时这个单元将从 fromRow 和 fromColumn 开始，扩展到 rowSpan 和 columnSpan 指定 的倍数的行和列。如果 rowSpan 或 columnSpan 的值为-1，则窗口部件将扩展到布局的底部 或者右边边缘处。

小贴士：栅格布局中的某个单元（cell）的长和宽，也可以说是栅格布局的行和列的尺寸 并不是一样大小的。如果你想使它们相等，你必须通过调用 setColumnMinimumWidth()和 setColumnStretch()方法来使得它们的最小宽度以及伸缩因子都彼此相等。

如果 QGridLayout 不是窗体的顶层布局（就是说它不能管理所有的区域和子窗口部 件），那么当你创建它的同时，就必须为它指定一个父布局，也就是把它加入到父布局中 去，并且在此之前，不要对它做任何的操作。使用 addLayout()方法可以完成这一动作。

在创建栅格布局完成后，就可以使用 addWidget()，addItem()，以及 addLayout()方 法向其中加入窗口部件，以及其它的布局。

当界面元素较为复杂时，应该毫不犹豫的尽量使用栅格布局，而不是使用水平和垂直布局的组合或者嵌套的形式，因为在多数情况下，后者往往会使 “局势”更加复杂而难以控 制。栅格布局赋予了界面设计器更大的自由度来排列组合界面元素，而仅仅带来了微小的复杂度开销。

当要设计的界面是一种类似于两列和若干行组成的形式时，使用表单布局要比栅格布 局更为方便些。

11.3.5 表单布局

表单布局是从 Qt 4.4 开始被引入的。它把布局内的界面元素分成两列，左边的列通常 放置标签（label）而右边的列放置对应值的窗口部件，如 line edits,spin boxes 等等, 如图 11-20 所示。表单布局在不同的平台上与本地原生外观相同，并且支持长的行形式，

如表 11-5 所示。

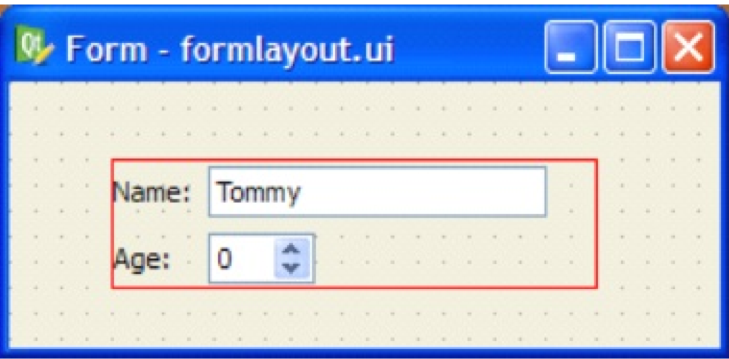


图 11-20 使用表单布局

QFormLayout 这个类是从 Qt 4.4 以后引入的。在使用该类之前，需要包含如下的头文 件声明。

```
#include &lt;QFormLayout&gt;;
```

在 QFormLayout 类出现以前，这种类似于两列布局的情形我们通常使用 QGridLayout 来创建。而现在，使用 QFormLayout 则有更多的优势：

能够适应不同的平台，并且提供与本地原生平台一致的观感

举例来说，在 Mac OS X Aqua 和 KDE 平台上通常要求标签中的文本是右对齐的，而在 Windows 和 GNOME 平台上则要求文本是左对齐的，QFormLayout 能够很好的自动适应。

支持可折叠的长行（当其中的文本较多时） Support for wrapping long rows. 在便携式设备上，文本太长以至于需要折叠的情况比较常见，但既要折叠而又要不影响显示的效果则是比较困难的事情。现在的 QFormLayout 类可以很好的解决这个问题，它会判断当前的情形是否需要折叠以及怎样折叠才好。表 11-5 显示了它的折叠策略。

表 11-5 QFormLayout 行折叠策略枚举值

常量	值	说明
QFormLayout::DontWrapRows	0	值域（即可输入的文本行）总是在其对应的标签(label)的旁边，也就是不 折叠。这是默认的行折叠策略（采用了 Qt 扩展风格的情况除外）。
QFormLayout::WrapLongRows	1	标签将获得最大允许的自由空间，与它对应的值域则使用剩余的空间，如 果一行放不下，它将另起一行，也就是把自己折叠起来。这是采用 Qt 扩展 风格情况的默认行折叠策略。
QFormLayout::WrapAllRows	2	值域将总是在与它对应的标签的下边一行。

API

为创建标签--（对应的）值域这种伙伴（buddy）类型的界面提供了丰富、方便的通过使用 addRow()函数（共有 6 种重载形式）或者 insertRow()函数（也有 6 种重载形式），我们可以简便的创建一个 QLabel 窗口部件以及它的伙伴窗口部件。一个实例代码 如下：

```
QFormLayout *formLayout = new QFormLayout;
formLayout-&gt;addRow(tr("&Name:"), nameLineEdit);
formLayout-&gt;addRow(tr("&Age:"), ageSpinBox);
setLayout(formLayout);
```

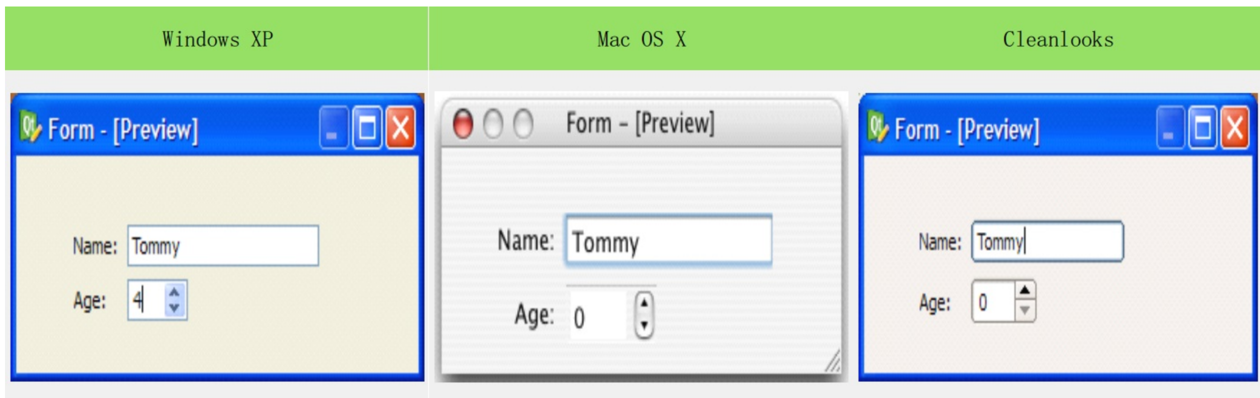
如果同样的情形使用 QGridLayout 来布局，则代码如下：

```
nameLabel = new QLabel(tr("&Name:"));
nameLabel-&gt;setBuddy(nameLineEdit);
ageLabel = new QLabel(tr("&Age:"));
ageLabel-&gt;setBuddy(ageSpinBox);
QGridLayout *gridLayout = new QGridLayout;
gridLayout-&gt;addWidget(nameLabel, 0, 0);
gridLayout-&gt;addWidget(nameLineEdit, 0, 1);
gridLayout-&gt;addWidget(ageLabel, 1, 0);
gridLayout-&gt;addWidget(ageSpinBox, 1, 1);
setLayout(gridLayout);
```

对比这两段代码就可以发现，它们实现了相同的界面，而使用 QFormLayout 类则更简单高效，并且不易出错。表 11-6 显示了该界面在不同的平台下的默认观感。

表 11-6 表单布局在不同平台上的默认外观

Windows XP	Mac OS X	Cleanlooks



小贴士：标签（QLabel）和它的伙伴（buddy）窗口部件

一个标签窗口部件和一个窗口部件具有伙伴关系，即指当用户按下激活标签的快捷键 时，鼠标/键盘的焦点将会移到它的伙伴窗口部件上。只有 QLabel 的实例才可以有伙伴窗 口部件，也只有该 QLabel 实例具有快捷键（在显示文本的某个字符前面添加一个前缀 &， 就可以定义快捷键）时，伙伴关系才有效。例如：

```
QLineEdit* nameLineEdit = new QLineEdit(this);
QLabel* nameLabel = new QLabel(tr("&Name"),this);
nameLabel-&gt;setBuddy(nameLineEdit);
```

该代码定义了 nameLabel 标签的快捷键为"Alt+P"，并将行编辑框 nameLineEdit 设置为它的伙伴窗口部件。所以当用户按下快捷键 "Alt+P"时，焦点将会跳至行编辑框 nameLineEdit 中。

在 Qt Designer 中，可以通过鼠标拖放操作快捷的建立 QLabel 和它的窗口部件的伙伴 关系，这需要切换到“伙伴编辑模式”。

由于 Qt 提供了许多的方法，所以使用表单布局时可以比较灵活的变换它的风格，经常 使用的方法有 setLabelAlignment()、setFormAlignment()、setFieldGrowthPolicy()、setRowWrapPolicy()等。举个例子，如果要在所有的平台上都模拟出 Mac OS X 上的观感，但却是用 Windows 上常见的标签文本左对齐的规则，可以这样写代码：

```
formLayout-&gt;setRowWrapPolicy(QFormLayout::DontWrapRows);
formLayout-&gt;setFieldGrowthPolicy(QFormLayout::FieldsStayAtSizeHint);
formLayout-&gt;setFormAlignment(Qt::AlignHCenter | Qt::AlignTop);
formLayout-&gt;setLabelAlignment(Qt::AlignLeft);
```

总结一下，创建表单布局的大致步骤如下：

第 1 步，创建布局内的窗口部件。

第 2 步，创建表单布局的实例，也就是创建一个 QFormLayout 类的实例，并将第 1 步 创建的窗口部件加入到布局之中。

第 3 步，调用 QWidget::setLayout()函数将布局安装到窗体上。

读者朋友可以通过前面的代码验证这些步骤。

11.3.6 删除布局内窗口部件的方法

要从一个布局内删除一个窗口部件，只需调用 QLayout::removeWidget()方法。其原型 如下：

```
void QLayout::removeWidget ( QWidget * widget )
```

这将删除该布局内的 widget 窗口部件，但是并没有把它从窗体界面上删除。调用完该函数后，你需要为该窗口部件指定一个合适的几何大小，或者干脆把它从界面上删除。一个 实例代码如下：

```
gridLayout-&gt;removeWidget(nameLabel);
nameLabel-&gt;setGeometry(9,9,50,25);
```

如果只是想使布局内的窗口部件隐藏起来，就可以调用 QWidget::hide()方法。然后调用 QWidget::show()方法可以使它再次显示。使用方法比较简单，读者可以自行验证。

如果往布局中添加一个窗口部件或者从布局中移除一个窗口部件，布局都会自动适应 所产生的这些新情况。如果对一个子窗口部件调用了 hide()或者 show()，也同样能够做到 自动适应。如果一个子窗口部件的大小提示发生了变化，布局将会自动进行调整，从而把新 的大小提示考虑进去。还有，布局管理器也会自动根据窗体中子窗口部件的最小大小提示和 大小提示，从总体上考虑，为这个窗体设置一个最小尺寸。

11.3.7 基本布局的综合运用

本实例利用基本布局管理（QHBoxLayout、QVBoxLayout、QGridLayout、QFormLayout）实现一个基于对话框的综合页面。实现效果如图 11-21 所示。实例代码见 basiclayouts 实例。

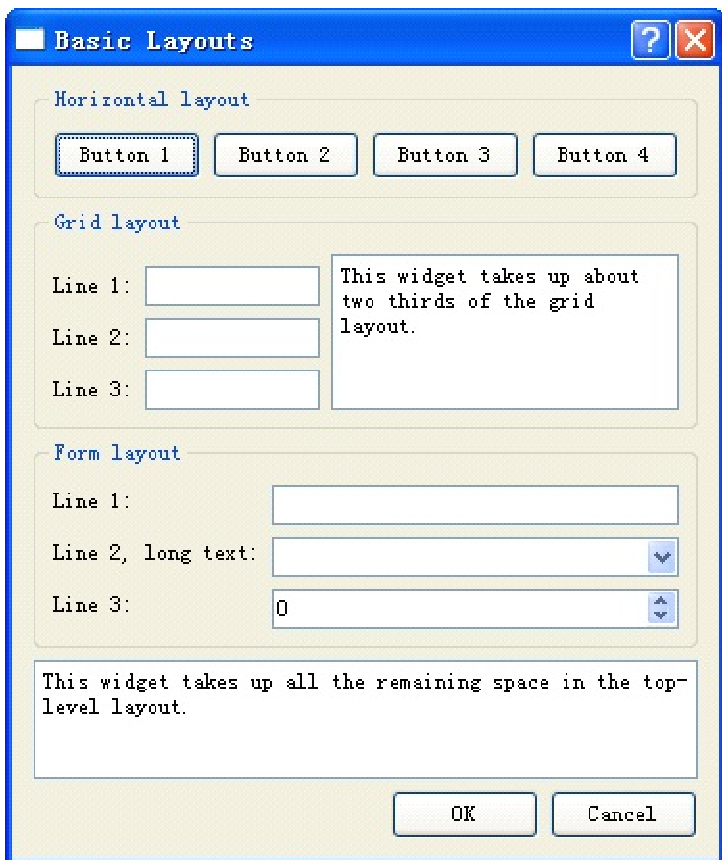


图 11-21 基本布局综合实例效果图

常用到的布局类有 QHBoxLayout、QVBoxLayout、QGridLayout、QFormLayout 这 4 种，分别是水平布局、垂直布局、栅格布局和表单布局。布局中最常用的方法是 addWidget()和 addLayout()，addWidget()方法用于在布局中插入窗口部件，addLayout()用于在布局中插 入子布局。对于表单布局而言，向其中加入窗口部件最常用的方法是 addRow()。

下面通过实例的实现过程了解布局管理的使用方法。首先通过一个示意图了解此对话 框的布局结构，如图 11-22 所示。

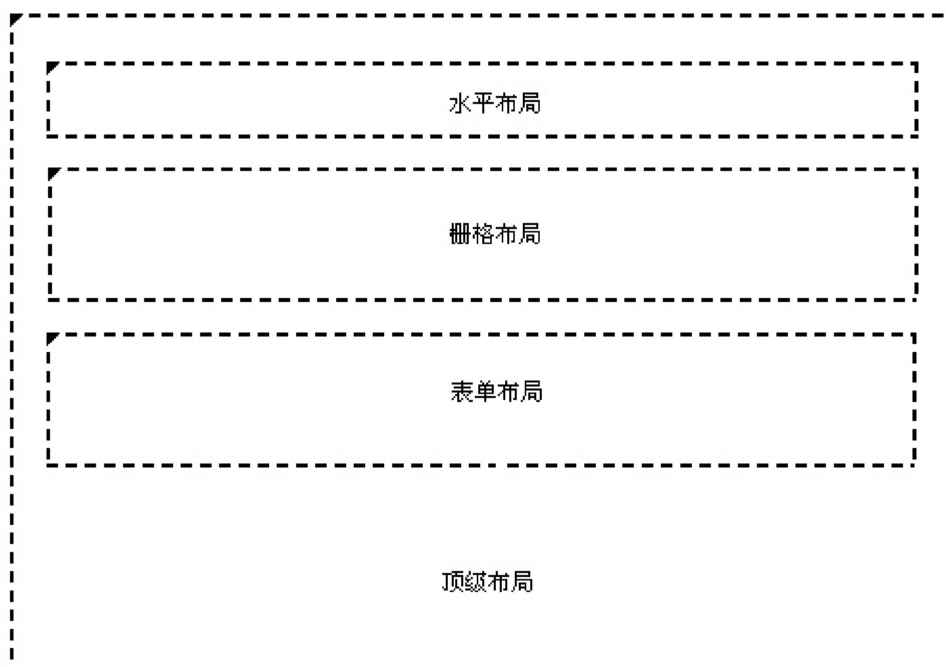


图 11-22 实例布局结构图

从图中可知，本实例共用到 4 个布局管理器，最外层也即是顶级布局是一个垂直布 局，它是 QVBoxLayout 的实例，在顶级布局里面的最上层是一个水平布局，它是 QHBoxLayout 的实例，在它的下面是一个栅格布局，它是 QGridLayout 的实例，然后下面是 一个表单布局，它是 QFormLayout 的实例。

下面是具体的实现，首先声明一个对话框类 Dialog，它单继承自 QDialog。在该类的 头文件 dialog.h 中声明对话框中的各个窗口部件。

```
class Dialog : public QDialog
{
    Q_OBJECT
public:
    Dialog();
private:
    void createHorizontalGroupBox();
    void createGridGroupBox();
    void createFormGroupBox();
    enum { NumGridRows = 3, NumButtons = 4 };
    QGroupBox *horizontalGroupBox;
    QGroupBox *gridGroupBox;
    QGroupBox *formGroupBox;
    QTextEdit *smallEditor;
    QTextEdit *bigEditor;
    QLabel *labels[NumGridRows];
    QLineEdit *lineEdits[NumGridRows];
    QPushButton *buttons[NumButtons];
    QDialogButtonBox *buttonBox;
};
```

然后定义对话框中包含的窗口部件。我们看看类的构造函数。

```
Dialog::Dialog()
{
    createHorizontalGroupBox();
    createGridGroupBox();
    createFormGroupBox();
    bigEditor = new QTextEdit;
    bigEditor-&gt;setPlainText(tr("This widget takes up all the remaining space "
        "in the top-level layout."));
    buttonBox = new QDialogButtonBox(QDialogButtonBox::Ok
        | QDialogButtonBox::Cancel);
    connect(buttonBox, SIGNAL(accepted()), this, SLOT(accept()));
    connect(buttonBox, SIGNAL(rejected()), this, SLOT(reject()));
    QVBoxLayout *mainLayout = new QVBoxLayout;
    mainLayout-&gt;addWidget(horizontalGroupBox);
    mainLayout-&gt;addWidget(gridGroupBox);
    mainLayout-&gt;addWidget(formGroupBox);
    mainLayout-&gt;addWidget(bigEditor);
    mainLayout-&gt;addWidget(buttonBox);
    setLayout(mainLayout);
    setWindowTitle(tr("Basic Layouts"));
}
```

第 3-5 行分别创建了 3 个组框-GroupBox，在它们的内部是 3 个布局。

第 6-7 行创建了创建了一个 QTextEdit 类的实例，并设置了它的显示文本。

第 8 行创建了一个 QDialogButtonBox 的实例，其中包含【OK】和【Cancel】两个按钮。

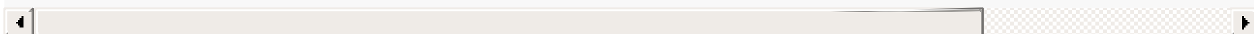
第 9-10 行连接按钮的信号和窗体的响应的槽。

第 11 行定义了窗体的顶级布局，它是一个 QVBoxLayout 的实例，是垂直布局。

第 12-16 行依次向顶级布局中加入窗口部件，注意使用 addWidget()方法的话，这些窗 口部件在布局中的排列顺序与它们的加入顺序是关联的。

小贴士：如果你想对布局内的窗口部件排列顺序和位置大小等属性做出更为详尽的控 制，可以考虑使用 insertWidget()方法来向布局中加入窗口部件。insertWidget()的原型 如下：

```
void QBoxLayout::insertWidget ( int index, QWidget * widget, int stretch = 0, Qt::Alignme
```



它为窗口部件 widget 指定了伸缩因子 stretch 和排列方向 alignment，以及在布局内 的顺序 index，然后把它加入到布局中。如果 index 的值为负整数，则表示把它加入到顺序 的最后面。

在布局中，哪个窗口部件的 stretch 值更大，那么它将有可能获得更大的空间。 如果在布局里面，所有窗口部件的 stretch 值都为 0，或者都不超过 0，那么它们的大小将由各自的大小策略以及大小策略互相影响的关系决定。

alignment 指定了窗口部件的排列方向，默认值为 0，表示该窗口部件将占用整个布局 内部的区域。alignment 通常有 3 个方向的取值，依次如如表 11-7、表 11-8 和表 11-9 所 示。

表 11-7 水平方向的取值

常量	值	说明
Qt::AlignLeft	0x0001	在布局内水平左对齐
Qt::AlignRight	0x0002	在布局内水平右对齐
Qt::AlignHCenter	0x0004	在可用的空间内部水平居中排列
Qt::AlignJustify	0x0008	Justifies the text in the available space.

注意，水平方向同时只能取一个值。

表 11-8 垂直方向的取值

常量	值	说明
Qt::AlignTop	0x0020	在布局内垂直向上对齐
Qt::AlignBottom	0x0040	在布局内垂直向下对齐
Qt::AlignVCenter	0x0080	在可用的空间内部垂直居中排列

垂直方向同时也只能取一个值。

表 11-9 居中取值

常量	值	说明
Qt::AlignCenter	AlignVCenter | AlignHCenter	在水平方向和垂直方向都居中排列

如果已近使用了居中取值的话，就不必再设置水平和垂直方向的取值了。

第 17 行将主布局安装到窗体上。

第 18 行设置窗口标题。

再来看看创建水平布局的 createHorizontalGroupBox()函数。

```
void Dialog::createHorizontalGroupBox()
{
    horizontalGroupBox = new QGroupBox(tr("Horizontal layout"));
    QHBoxLayout *layout = new QHBoxLayout;
    for (int i = 0; i &lt; NumButtons; ++i)
    {
        buttons[i] = new QPushButton(tr("Button %1").arg(i + 1));
        layout-&gt;addWidget(buttons[i]);
    }
    horizontalGroupBox-&gt;setLayout(layout);
}
```

第 4 行定义了一个水平布局的实例。

第 5-9 行定义了几个按钮窗口部件，然后依次向该布局中加入它们。

第 10 行把该布局安装到它的父窗体上。 然后看看创建栅格布局的函数 createGridGroupBox()里面的内容。

```
void Dialog::createGridGroupBox()
{
    gridGroupBox = new QGroupBox(tr("Grid layout"));
    QGridLayout *layout = new QGridLayout;
    for (int i = 0; i &lt; NumGridRows; ++i)
    {
        labels[i] = new QLabel(tr("Line %1:").arg(i + 1));
        lineEdits[i] = new QLineEdit;
        layout-&gt;addWidget(labels[i], i + 1, 0);
        layout-&gt;addWidget(lineEdits[i], i + 1, 1);
    }
    smallEditor = new QTextEdit;
    smallEditor-&gt;setPlainText(tr("This widget takes up about two thirds of the "
        "grid layout."));
    layout-&gt;addWidget(smallEditor, 0, 2, 4, 1);
    layout-&gt;setColumnStretch(1, 10);
    layout-&gt;setColumnStretch(2, 20);
    gridGroupBox-&gt;setLayout(layout);
}
```

第 1 行创建了一个组框的实例，它将作为栅格布局的父窗口。

第 3-9 行创建了若干 QLineEdit 的实例，并使用 addWidget()方法把它们加入到布局 中。addWidget()有若干变型，这里使用的这个需要依次指定窗口部件、行和列以及对齐方 式，对齐方式默认为 0。注意可以看到这里的第 1 个 QLineEdit 实例位于(1,0)这个单元 (cell)。

第 10-11 行创建一个 QTextEdit 实例，并为它设置显示文本。

第 12 行将 QTextEdit 的实例 smallEditor 加入到布局中，这里使用的是 addWidget() 的另一种常用变型，其实参分别需要指定窗口部件、起始行、起始列、行跨度数和列跨度 数。

在本实例中实际上创建了一个 3x3 的栅格布局。左上角的文本为 Line 1 的标签窗口部 件位于(1,0)，这样就可知道使用本行创建的 smallEditor 窗口部件位于(0,2)，行跨度是 4 行，列跨度是 1 列，也就是它的单元占用了 4 行 1 列的空间。

第 13-14 行分别设置了第 1 列和第 2 列的伸缩因子为 10 和 20，这就保证了这两列无论 何时均保持宽度为 1:2。

第 15 行把这个栅格布局安装到其父窗口 gridGroupBox 上。 再看一下创建表单布局的 createFormGroupBox()函数内容。

```
void Dialog::createFormGroupBox()
{
    formGroupBox = new QGroupBox(tr("Form layout"));
    QFormLayout *layout = new QFormLayout;
    layout-&gt;addRow(new QLabel(tr("Line 1:")), new QLineEdit);
    layout-&gt;addRow(new QLabel(tr("Line 2, long text:")), new QComboBox);
    layout-&gt;addRow(new QLabel(tr("Line 3:")), new QSpinBox);
    formGroupBox-&gt;setLayout(layout);
}
```

第 1 行创建了一个组框，用于管理表单布局。

第 2 行创建了一个表单布局的实例。

第 3-5 行使用 addRow()方法，创建了几个 QLabel 的实例和 QLineEdit、QComboBox 和 QSpinBox 的实例，并把它们加入到表单布局中。在前面讲解表单布局的时候，曾经说到过 addRow()方法有很多种变型，此处采用的是其中的一种。

第 6 行把表单布局安装到它的父窗口上，它的父窗口就是第 1 行创建的组框实例。 最后书写主函数 main.cpp，其内容是创建应用程序全局实例，将刚才创建的窗口显示出来。

```
int main(int argc, char *argv[])
{
    QApplication app(argc, argv);
    Dialog dialog;
    return dialog.exec();
}
```

这个实例完全采用手写代码实现。可以自己书写 .pro 文件，也可以在命令行下依次键 入 qmake 来生成。

本实例分析了 Qt4 中布局管理常用到的类及其方法，如果读者觉得这种手动布局的方 法比较麻烦，也可采用 Qt Designer 来布局。

11.4 堆栈布局

堆栈布局也是布局管理器的一种。它把一系列的窗口部件排列成类似堆栈的样子，但 每次只能有一个窗口部件是当前的可见窗口。

11.4.1 使用方法

QStackedLayout 类被用来创建堆栈布局的实例。使用堆栈布局创建出来的应用程序的 样子与使用 QTabWidget 创建的效果有些相像。也可以使用 QStackedWidget 类来创建一个 应用程序界面，和使用 QStackedLayout 效果差不多，目前在最新的 4.5.2 版 Qt 中，我们如果使用 Qt Designer 创建用户界面，那么通常会使用 QStackedWidget，这点在后面还会讲到。

实际上，你可以把堆栈布局看成是由一系列的子窗口部件组成的，它们都是一些“页面”（pages）。

在使用堆栈布局时，首先应包含其头文件：

```
#include &lt;QStackedLayout&gt;
```

下面是手写代码创建一个堆栈布局的实例：

```
QWidget *firstPageWidget = new QWidget;
QWidget *secondPageWidget = new QWidget;
QWidget *thirdPageWidget = new QWidget;
QStackedLayout *stackedLayout = new QStackedLayout;
stackedLayout-&gt;addWidget(firstPageWidget);
stackedLayout-&gt;addWidget(secondPageWidget);
stackedLayout-&gt;addWidget(thirdPageWidget);
QVBoxLayout *mainLayout = new QVBoxLayout;
mainLayout-&gt;addLayout(stackedLayout);
setLayout(mainLayout);
```

创建栈布局的大致步骤如下：

第 1 步，创建布局内的窗口部件，也即是各个页面。

第 2 步，创建栈布局的实例，也就是创建一个 QStackedLayout 类的实例，并将第 1 步 创建的窗口部件加入到布局之中。

向布局内加入窗口部件的方法可以使用 addWidget()，也可以使用 insertWidget()， 前者将把窗口部件加入到子窗口索引的最后，而后者则可以指定在索引中的位置加入窗口部 件。可以根据实际需要选用。

第 3 步，调用 QWidget::setLayout()函数将布局安装到窗体上。 11.4.2 如何索引窗口部件

由于 QStackedLayout 类并没有提供位于其内部的窗口部件（即上文提到的 “页面”） 的索引，所以我们在使用堆栈布局时，通常需要使用组合框类（ QComboBox）或者列表部件 类（QListWidget）的实例来存储这些子窗口的索引，继而实现切换堆栈布局中子窗口的目 的。下面是一个使用 QComboBox 类的实例：

```
QComboBox *pageComboBox = new QComboBox;
pageComboBox-&gt;addItem(tr("Page 1"));
pageComboBox-&gt;addItem(tr("Page 2"));
pageComboBox-&gt;addItem(tr("Page 3"));
connect(pageComboBox, SIGNAL(activated(int)),
stackedLayout, SLOT(setCurrentIndex(int)));
```

在上述代码中，通过 addItem()方法为 QComboBox 的实例依次加入堆栈布局的索引，这样当调用 connect()函数显式的连接 pageComboBox 的 activated()信号和 stackedLayout 的 setCurrentIndex()槽时，这些索引就会被加入到内部的一个链表中，这样就可以方便的 查找布局内的窗口部件了。

要获得栈布局内的窗口部件的数量，可以使用 count()方法。 那么我们如何找到位于堆栈布局内的任意一个子窗口呢，可以使用堆栈窗口的 widget() 方法返回当前指定位置索引的子窗口，它的原型如下：

```
QWidget * QStackedLayout::widget ( int index ) const
```

它将返回由 index 指定的位置的子窗口，当返回值为 0 时，则表示在 index 指定的位置处并没有任何子窗口。

作为上述的一个特例，要找到当前的活动子窗口，可以先找到当前它对应的 index，方 法是使用 currentIndex()方法，它的原型如下：

```
int currentIndex () const
```

如果要取得某一个窗口部件在布局中的索引，可以使用 indexof()方法，其原型如下：

```
int QLayout::indexOf ( QWidget * widget ) const [virtual]
```

该方法将 widget 在布局内的索引返回，如果返回值为-1，则表示没有找到这个窗口部件。

小贴士：实际上堆栈布局有一个重要的属性 currentIndex : int，是一个 int 常量，它对 应当前活动子窗口的索引，获得该属性值的方法即是调用 currentIndex()方法。如果它的 值为-1，则表示没有当前活动子窗口，也就是没有子窗口在堆栈布局内。

要想找到当前的活动子窗口，可以使用 currentWidget()方法，它的原型如下：

```
QWidget * QStackedLayout::currentWidget () const
```

它将返回堆栈布局内当前的活动子窗口，如果返回值为 0，则表示没有当前的 堆栈布局内没有子窗口，由此这个函数也可以被用来判断当前布局内是否存在窗口部件。

要想指定一个位于堆栈布局内部的子窗口为当前的活动窗口，可以使用 setCurrentWidget()方法，它的原型如下：

```
void QStackedLayout::setCurrentWidget ( QWidget * widget ) [slot]
```

它将把你指定的 widget 作为当前的活动子窗口，前提是这个子窗口必须已经位于堆栈 布局内部。

此外，一旦堆栈布局内当前的窗口部件产生了变化或者被移除，布局就会发 出 currentChanged() 以及 widgetRemoved() 信号。

11.4.3 实例-堆栈窗体

本实例实现一个堆栈窗体的使用，实现的效果如图 11-23 所示。当用户选择左侧列表 框中不同的选项时，右侧则对应显示所选的窗体。

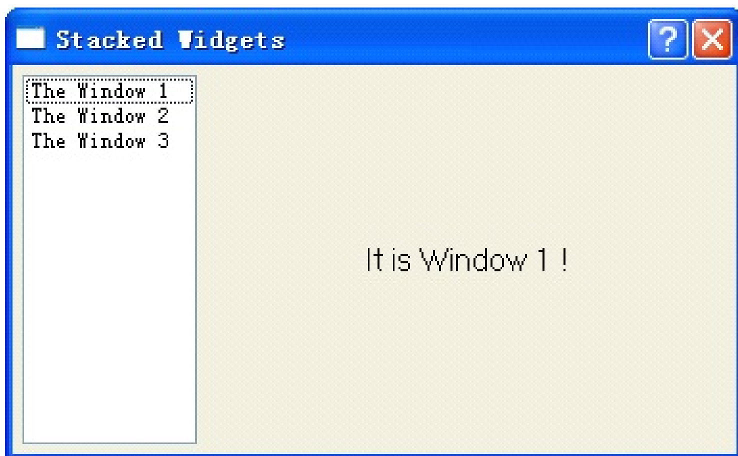


图 11-23 堆栈窗体实例

为了使读者朋友熟练掌握使用方法，这里将用两种方法为大家分别实现，一是完全手 写代码的方法，源代码见实例 stack。二是使用 Qt Designer 设计界面，然后在 Qt Creator 中创建工程辅以部分手写代码的方法，源代码见 stackDlg。

1.完全手写代码

实现头文件 stackdlg.h。

```
class StackDlg : public QDialog
{
    Q_OBJECT
    public:
    StackDlg(QWidget *parent = 0, Qt::WindowFlags f1 = 0);
    QLabel *label1;
    QLabel *label2;
    QLabel *label3;
    QListWidget *listWidget;
    QStackedWidget *stackWidget;
};
```

在头文件中声明一个对话框类，它继承自 QDialog，然后声明所用到的窗口部件。 实现源文件 stackdlg.cpp。

```
StackDlg::StackDlg(QWidget *parent, Qt::WindowFlags f1)
: QDialog(parent,f1)
{
    setWindowTitle(tr("Stacked Widgets"));
    listWidget = new QListWidget(this);
    listWidget-&gt;insertItem(0,tr("The Window 1"));
    listWidget-&gt;insertItem(1,tr("The Window 2"));
    listWidget-&gt;insertItem(2,tr("The Window 3"));
    label1 = new QLabel(tr("It is Window 1 !"));
    label2 = new QLabel(tr("It is Window 2 !"));
    label3 = new QLabel(tr("It is Window 3 !"));
    stackWidget = new QStackedWidget(this);
    stackWidget-&gt;addWidget(label1);
    stackWidget-&gt;addWidget(label2);
    stackWidget-&gt;addWidget(label3);
    QHBoxLayout *mainLayout = new QHBoxLayout(this);
    mainLayout-&gt;setMargin(5);
    mainLayout-&gt;setSpacing(5);
    mainLayout-&gt;addWidget(listWidget);
    mainLayout-&gt;addWidget(stackWidget,0,Qt::AlignHCenter);
    mainLayout-&gt;setStretchFactor(listWidget,1);
    mainLayout-&gt;setStretchFactor(stackWidget,3);
    connect(listWidget,SIGNAL(currentRowChanged(int)),stackWidget,SLOT(setCurrentIndex(in
}
```

第 1 行设置应用程序的标题，注意要使用 tr()函数。

第 2-5 行创建乐一个 QListWidget 窗口部件的实例，并在其中插入 3 个条目，当我们

在 3 个条目之间切换时，右面的堆栈窗口将会对应变化 。

第 6-8 行依次创建了 3 个 QLabel 窗口部件的实例，作为右面的堆栈窗口中对应显示的 3 个窗体。

第 9 行创建了一个 QStackWidget 堆栈窗体的实例。

第 10-12 行调用 addWidget()方法把前面创建的 3 个 QLabel 窗体部件的实例依次插入 到堆栈窗中。

第 13 行设置了一个顶层（Top Level）布局，它是一个 QHBoxLayout 实例。 第 14 行调用 setMargin()函数设置布局距离窗体边界的尺寸。

第 15 行调用 setSpacing()设置了在该布局中的窗体部件之间的距离。该函数的原型如 下：

```
void setSpacing ( int )
```

小贴士：QLayout 有两个非常重要的属性：

- sizeConstraint : SizeConstraint 该属性定义了布局的伸缩模式。
- spacing : int 该属性定义了在一个布局内部的窗口部件之间的距离，也就是间距。

如果没有明确的设置该属性值，那么该布局将继承它的父布局的设置，或者该布局中的 窗口部件将继承它们的父窗口部件的设置。

再次提醒，当使用 QGridLayout 或者 QFormLayout 布局时，很可能要分别设置不同的水 平和垂直方向的间距，这时就可以使用 setHorizontalSpacing() 、 setVerticalSpacing()这两个方法。当调用了这两个方法设置了水平和垂直间距后，再调用 spacing()获取该属性，这时它的返回值就是-1 了。

第 20 行连接 QListWidget 的 currentRowChanged()信号与堆栈窗的 setCurrentIndex() 槽，实现按选择显示窗体。此处的堆栈窗体 index 按插入的顺序从 0 起依次排序，与 QListWidget 的条目排序相一致。

2.Qt Designer 结合 Qt Creator 实现

本实例分析了堆栈窗的基本使用方法。在实际应用中，堆栈窗口多与列表框 QListWidget 或者下拉列表框 QComboBox 配合使用。

这种方法是前面第 5 章所讲述的 Qt 编程的 3 种基本方法之一，在此引领大家温习一 遍。对于.ui 文件的使用我们选用单继承法，如果对此法的使用还有不熟悉的地方，请回到 第 7 章复习。

项目的实现步骤如下：

第 1 步，创建一个新的项目

打开 Qt Designer，创建一个名为 stackDesigner 的新的项目，项目类型为用以存放项目文件。

第 2 步，创建程序界面文件（.ui 文件）

I. 启动 Qt Designer，在其中创建本程序的界面，模板的类型可以使 Widget 或者是 Dialog 等。

II. 从窗口部件盒内依次拖拉出 1 个 ListWidget、1 个 Horizontal Spacer 和 1 个 Stacked Widget,把它们摆放在大致的位置即可，它们构成了顶级布局的要素。

III. 然后在 ListWidget 的界面上双击鼠标左键，即可在弹出的如图 11-24 所示的【编辑 列表窗口部件】对话框中为列表添加项目，依次添加 3 个：window1、window2 和 window3。



图 11-24 为列表窗口部件添加项目

IV. 接下来，为堆栈窗口部件添加子窗口部件，也就是页。初始情况下，堆栈窗口已经 为我们内置了两个页，我们从窗口部件盒中拖出 2 个 Label，分别放入两个页中。

注意，切换这两个页的方式是用鼠标点击堆栈窗口右上角的那两个黑色的三角形的导 航按钮，左边的那个是向前翻页，右边的那个是向后翻页。

V. 然后我们需要为堆栈窗口部件添加第 3 个页，方法先切换到第 2 个页，然后用鼠标 右键点击那两个导航按钮之一，在弹出的上下文菜单中依次选择 【插入页】→【在当前页 之后】，就完成了第 3 个页的插入。然后仿照上面的步骤，向其中加入 1 个 Label。

VI. 这之后，再在每个页面上 Label 的左右两侧各放置一个 Horizontal Spacer。

VII. 接下来我们为窗口部件设置属性，为简单起见，都取默认属性，而 3 个 Label 窗口 部件的 text 属性设置为 It is Window 1、It is Window 2 和 It is Window3。

必须要设置的是 Stacked Widget 的第 3 个页的 objectName 属性，这是 Qt Designer 的一个 bug，在目前的 4.5.2 版 Qt 里，你自己增加的页面的 objectName 属性将被 Qt Designer 命名为类似“页”的乱码，这将导致后面 qmake 运行时的错误。把它手动修改为 page_3。

VIII. 然后我们从里到外，依次为窗口部件布置布局。

布置好的样子如图 11-25 所示，你可以清楚的看到各个布局的情况，在此不再赘述。

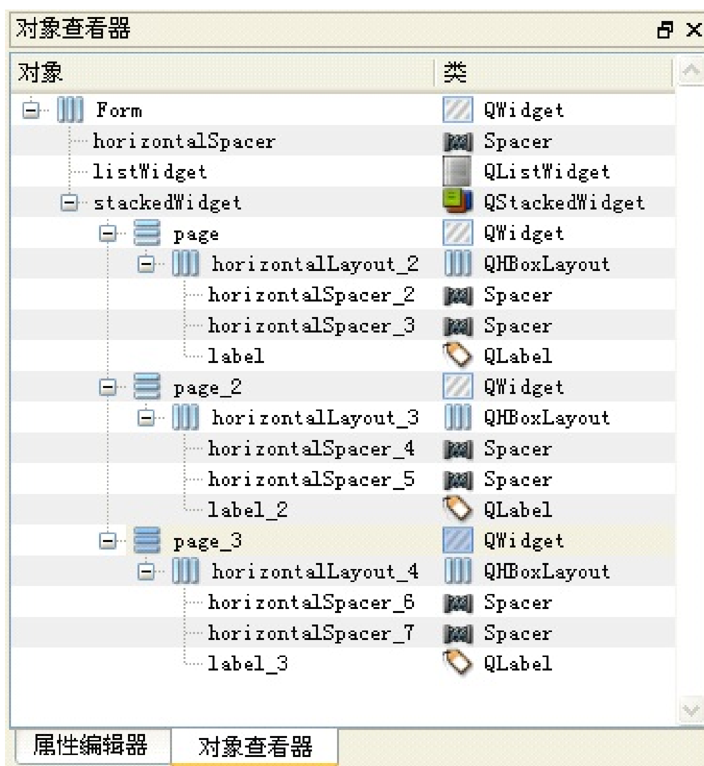


图 11-25 在对象查看器中看到的布局情况

第 3 步，向项目中引入该界面文件。

把这个文件命名为 stack Designer，保存在我们的项目目录下。在项目上点击鼠标右 键，在上下文菜单中选择【Add Existing Files...】，把该.ui 文件加入到工程中。

第 4 步，运行 qmake，以生成 ui_stackDesigner.h 头文件。

这时，先运行一下 qmake，它将调用 moc，以生成需要的 ui_stackDesigner.h 头文件。

第 5 步，自定义一个界面类，采用单继承方法使用界面文件。

首先新建一个名为 stackDesignerDlg.h 的头文件，其内容如下：

```
#ifndef STACKDESIGNERDLG_H
#define STACKDESIGNERDLG_H
#include &lt;QtGui&gt;
#include "ui_stackDesigner.h"
class StackDesignerDlg : public QDialog
{
    Q_OBJECT
public:
    StackDesignerDlg(QWidget *parent = 0, Qt::WindowFlags f1 = 0);
private:
    Ui::Form ui;
};
#endif // STACKDESIGNERDLG_H
```

在其中定义了该类公有单继承自 QDialog，以及它的构造函数。最重要的是声明了一个私有变量 ui，它是界面原生类的对象，这是单继承法最明显的特征。再建立该类的实现文件，其内容如下：

```
#include "stackDesignerDlg.h"
StackDesignerDlg::StackDesignerDlg(QWidget *parent, Qt::WindowFlags f1)
: QDialog(parent,f1)
{
    setWindowTitle(tr("Stacked Widgets"));
    ui.setupUi(this);
    connect(ui.listWidget,SIGNAL(currentRowChanged(int)),ui.stackedWidget,SLOT(setCurrent
}
```

为简单起见，这里主要实现了其构造函数。

首先加入了类头文件声明。然后设置程序的标题，调用 setupUi()方法来完成界面布局 的初始化。

接下来把 listWidget 的 currentRowChanged()信号和 stackedWidget 的 setCurrentIndex()槽关联起来。

第 6 步，书写主程序文件。

完成了界面类的定义后，最后新建一个 main.cpp 文件，其内容如下：

```
#include &lt;QApplication&gt;
#include "stackDesignerDlg.h"
int main( int argc, char * argv[] )
{
    QApplication a( argc, argv );
    StackDesignerDlg stack;
    stack.show();
    return a.exec();
}
```

这段代码中定义一个我们刚才定义的界面类的实例，然后把它显示出来。

第 7 步，编译运行程序。

我们采用快捷键，依次按下 Ctrl+B 和 Ctrl+R 编译运行程序，其效果如图 11-26 所 示，与手写代码无异。

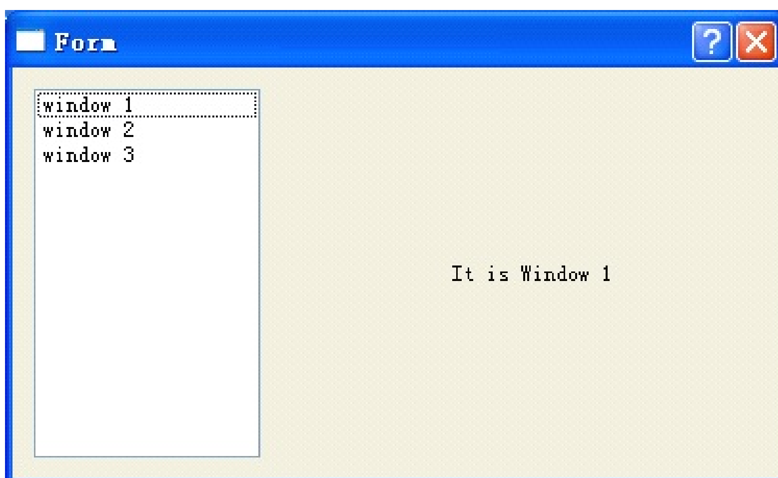


图 11-26 程序运行效果

需要注意的是，目前的 Qt Creator 并没有把 Qt Designer 的全部功能集成进来，比如 我们无法在其中预览 Qt Designer 生成的界面效果，因而笔者建议大家最好还是直接在 Qt Designer 中设计界面，然后使用 Qt Creator 完成项目的构建。

总而言之，这两种方法都是常用的，且各有长处。手写代码的方式似乎更加简洁，而 Qt Designer 结合 Qt Creator 的方式能使编程过程简化，且可以直观的验证你的布局情 况。读者朋友可以根据自己的喜好掌握。而笔者建议最好是将手写代码方式弄通，熟练之后 再使用 IDE 的方式，这样也能更好的理解 Qt 编程的精髓。

11.5 分裂器布局

QSplitter 实质上是一个窗口部件，但同时它可以包含一些其他窗口部件。在切分窗口（splitter）中的这些窗口部件会通过切分条（splitter handle）而分隔开来。用户可以 通过拖动这些切分条来改变切分窗口中子窗口部件的尺寸。切分窗口常常可以用作布局管理 器的替代，从而可以把更多的控制权交给用户。

11.5.1使用方法

大家知道，QSplitter 是一个容器类，Qt Designer 把分裂器对象视为可以容纳其它窗 口部件的布局。在 Qt Designer 中要使用分裂器布局也很容易，如图 11-27 所示，先选中 要布局的界面元素，然后选择工具栏上对应的按钮或者通过菜单项或者鼠标右键的上下文菜 单就可以完成。

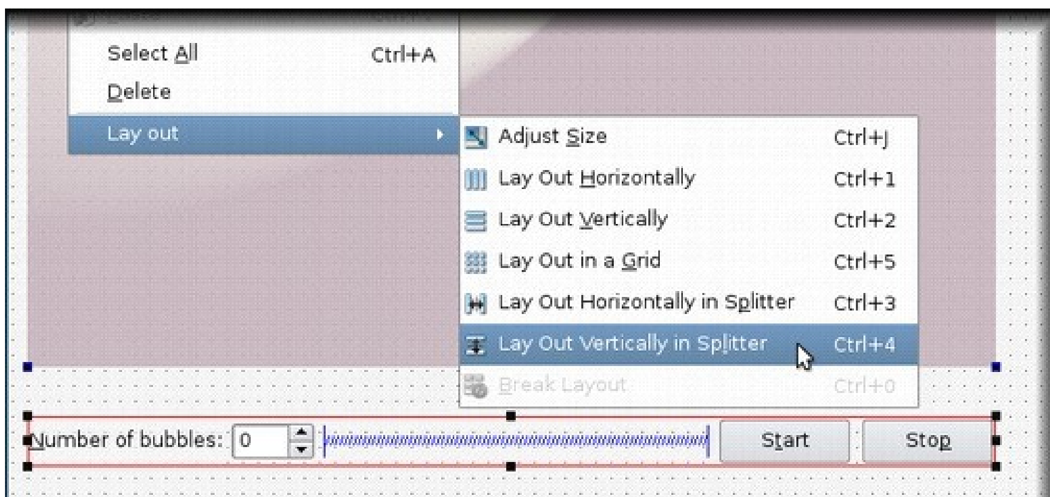


图 11-27 使用分裂器布局

QSplitter 类可以用来创建分裂器布局，继而实现切分窗口。 创建一个分裂器布局的一般步骤如下：

第 1 步，创建要使用的窗口部件。

第 2 步，创建分裂器布局的实例，就是 QSplitter 的实例。

第 3 步，使用 insertWidget()或者 addWidget()方法把第 1 步创建的窗口部件加入到 布局之中。

第 4 步，调用 QWidget::setLayou()方法把布局安装到窗体上。 注意，在使用分裂器布局之前，需要包含它的头文件声明：

```
#include &lt;QSplitter&gt;
```

11.5.2 构造函数

QSplitter 类有两个构造函数的原型：

```
QSplitter::QSplitter ( Qt::Orientation orientation, QWidget * parent = 0 )
```

参数 orientation 指定了分裂器的方向是水平的还是垂直的， parent 指定了父窗口。一 个使用该型构造函数的代码示例如下：

```
QSplitter *splitterRight = new QSplitter(Qt::Vertical,splitterMain);
```

该代码定义了一个垂直分裂器布局，并指定了它的父窗口为 splitterMain，后者也是一个分裂器布局。

```
QSplitter::QSplitter ( QWidget * parent = 0 )
```

这个型别实质上是第一种构造函数的缺省变体，它默认创建一个水平的分裂器布局。 一个使用该型构造函数的代码示例如下：

```
QSplitter splitter(0);
```

该代码定义了一个水平分裂器布局，并且采用程序上下文中的窗口作为父窗口。

这两种构造函数型我们都会经常用到，请读者朋友注意掌握。

11.5.3 一些深入的话题

默认情况下，在分裂器布局中的窗口部件会尽可能的按照用户的意愿在它的最小大小 提示以及最小大小或者是最大大小之间调整。如果有需要，也可以调用 QSplitter 类的 setSizes()方法来为布局中每个窗口部件指定大小。而要获取分裂器布局内当前的各个窗口 部件的大小，可以使用 QSplitter 类的 sizes()方法。

如果你想保存分裂器布局的构造，可以使用 QSplitter 的 saveState()和 restoreState()方法。它们通常与 QSettings 类结合使用，保存设置的代码如下：

```
QSettings settings;
settings.setValue("splitterSizes", splitter-&gt;saveState());
```

恢复设置的代码如下：

```
QSettings settings;
splitter-&gt;restoreState(settings.value("splitterSizes").toByteArray());
```

同前面讲过的堆栈布局的情形类似，如果你想获取分裂器布局中的窗口部件的相关信息，可以通过调用 indexof()、widget()以及 count()等方法来实现。

当你调用 hide()方法隐藏了分裂器布局中的某个窗口部件时，它会在界面上消失掉， 并且它原先占有的空间将被其它的窗口部件所 “分享”。而一旦你调用 show()方法显示它 时，一切又会恢复原样。

分裂器布局也是经常使用的一种，它分为分裂器水平布局和分裂器垂直布局，这可以 通过设置分裂器布局的方向来确定，如 Qt::Horizontal 和 Qt::Vertical。使用它们时，界 面效果和使用常见的水平和垂直布局几乎没有区别，它的最明显特征是布局内的元素之间是 等间距的，而水平和垂直布局则不一定是这样。

11.5.4 分裂器布局实例

在 QSplitter 中的子窗口部件将会自动按照创建时的顺序一个挨一个的（或者一个在 另外一个的下面）放在一起，并以切分窗口拖动条（ splitter bar）来分隔相邻的窗口部 件。以下是用于创建如图 11-28 所示的分裂器水平布局的窗口的代码。完整的源代码见实 例 splitter。

```
#include &lt;QtGui&gt;
int main(int argc, char *argv[])
{
    QApplication app(argc, argv);
    QListWidget *listWidget = new QListWidget;
    QTreeWidget *treeWidget = new QTreeWidget;
    QTextEdit *editor = new QTextEdit;
    QSplitter splitter(Qt::Horizontal);
    splitter.addWidget(listWidget);
    splitter.addWidget(treeWidget);
    splitter.addWidget(editor);
    listWidget-&gt;addItem(QObject::tr("Inbox"));
    listWidget-&gt;addItem(QObject::tr("Outbox"));
    listWidget-&gt;addItem(QObject::tr("Sent"));
    listWidget-&gt;addItem(QObject::tr("Trash"));
    treeWidget-&gt;setColumnCount(1);
    QList&lt;QTreeWidgetItem *&gt; items;
    for (int i = 0; i &lt; 10; ++i)
        items.append(new QTreeWidgetItem((QTreeWidget*)0, QStringList(QString("item:%1").
    treeWidget-&gt;insertTopLevelItems(0, items);
    editor-&gt;setPlainText(QObject::tr("My child, my sister,\n"
        "think of the sweetness\n"
        "of going there to live together!\n"
        "To love at leisure,\n"
        "to love and to die\n"
        "in a country that is the image of you!"));
    splitter.setWindowTitle(QObject::tr("Splitter"));
    splitter.show();
    return app.exec();
}
```

把这段代码保存成一个.cpp 文件，如 splitter.cpp,并创建一个文件夹 splitter，把splitter.cpp 文件放入其中。依次运行 qmake –project，qmake splitter.pro,mingw32- make，即可生成可执行文件。程序的运行效果如图 11-28 所示。

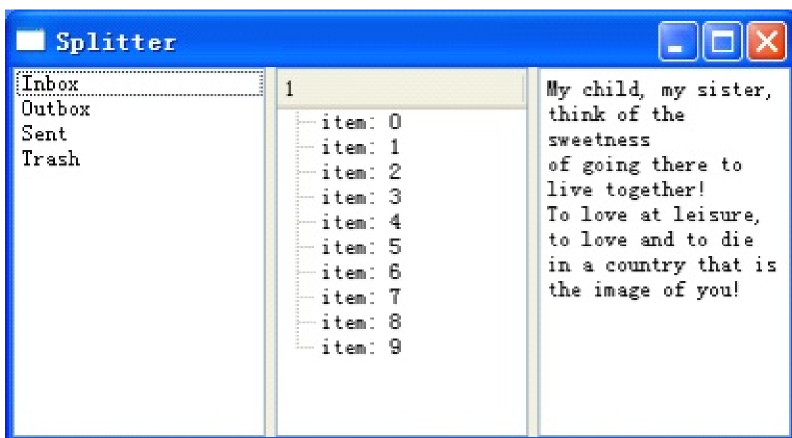


图 11-28 分裂器布局实例效果

该程序代码比较易懂，我们简要的讲解一下。

第 1 行，加入了程序中用到的头文件声明，这也包括了 QSplitter 类的声明在里面。 第 5-7 行定义了布局内用到的窗口部件。

第 8 行定义了一个分裂器水平布局。

第 9-11 行将窗口部件加入到分裂器布局之中。

第 12-15 行为 listWidget 添加一些项目。注意在这个程序中，由于 main()主函数并不 是属于某个 QObject 类的子类，所以不能直接使用 tr()函数，而需要静态调用它，即 QObject::tr()。

第 16 行设置 treeWidget 的列数为 1。 第 17 行声明一个链表对象 items。

第 18-19 行为 items 赋值。

第 20 行将 items 的值加入到 treeWidget 中。 第 21 行为 editor 设置文本。

第 22 行设置窗口标题。

第 23-24 行显示窗体。

11.6 自定义布局管理器

要使用自定义的布局管理器，我们可以重新实现 addItem(), sizeHint(), setGeometry(), itemAt() 和 takeAt()这些方法。为了确保当应用程序界面的空间非常小时，布局大小不会为0，我们需要重载 minimumSize()方 法。日常中我们也经常看到这样的情形，即应用程序窗口的长和宽的尺寸是互为依存的，那 么我们就需要重载 hasHeightForWidth()和 heightForWidth()方法来实现这种效果。

这部分内容已经超出了本书的范围，感兴趣的读者请查阅 Qt Assistant 中的 Layout Classes 条目。

11.7 布局管理经验总结

好了，到了现在，是时候为布局管理这个话题做一下总结了。

在大多数情况下，Qt 的布局管理器将为管理的窗口部件选择最优尺寸，以便窗口可以 顺利的重新调整大小。如果默认值不合理，那么 我们可以使用以下方法优化布局：

1. 为某些子窗口部件设置最小尺寸、最大尺寸或固定尺寸

2. 添加伸缩项目（stretch）或间距项目 这些项目将填补布局中的空白区域。手写代码即是调用 addStretch()，在 Qt Designer 中则是通过添加一个 spacer 窗口部件来实现。

3. 更改子窗口部件的大小策略

通过调用 QWidget::setSizePolicy()，编程人员可以采用最优的方式重新设置子窗口 部件的尺寸变化行为。可以根据布局中其他子窗口部件来扩大、缩小子窗口部件，或者使其 尺寸不变。

4. 更改子窗口部件的大小提示

QWidget::sizeHint() 和 QWidget::minimum SizeHint() 可以根据窗口部件的内容返 回其首选尺寸和首选最小尺寸。Qt 内建的窗口部件已经相应的提供了合适的实现。

5. 设置伸缩因子

伸缩因子支持子窗口部件的相对增长；例如，将 2/3 的任何多余的可用空间分配给 A 窗口部件，将 1/3 的空间分配给 B 窗口部件，这将使得两者的比例保持在 2:1 的比例 上。

6.设置被布局管理的窗口部件之间的“间距”和整个布局周围的“空白”

默认情况下，Qt 使用与上下文相关的行业标准值。

7.自定义布局管理器

当 Qt 内建的这些布局管理器都不能很好的满足你的需要时，就可以考虑自定义布局管 理器。当然根据我的经验，这种情况是比较少见的。

8.各种布局方法综合使用 在有些情况下，可以将绝对位置法、人工布局法与布局管理器结合使用，往往会收到很好的效果。

当布局管理器在摆放这些窗口部件的时候，它就会考虑到上面讲到的这些约束条件。 并且如果这些还不满足你的要求的话，就可以对子窗口部件的类进行派生并且重新实现 sizeHint()函数，由此获得所需的大小提示。

11.8 问题与解答

问：关于 layout 布局与控件的大小

尽管我把一个窗口部件的 size 高和宽都设为 fixed，但是每次选用 layout 功能时，窗 口部件的 size 又会发生变化，有什么办法让窗口部件的大小一直维持在自己定义的大小， 无论外界布局做任何变化都不受影响？

答：与这个问题相关的内容很多，就是说布局受影响的因素很多。比如几种大小策略 在一起时的优先问题，某个大小策略的 “倾向性”问题，布局内部的各个窗口部件有没有设 置伸缩因子等问题，等等。通常你首先需要设置窗体的顶级布局，然后为窗体内的窗口部件 设置大小策略、伸缩因子、最大大小、最小大小等。另外，在需要绘制比较复杂的布局时， 手写代码的控制能力通常更强一些。

问：请教 Qt 布局的问题

最近发现在代码中手动建立的控件和布局 ，在窗口最大化的时候就可以自动调整大小，但是在 Qt desinger 里面画好的界面，虽然也加了布局，但是在窗口最大化的时候却 无法自动调整控件的大小，以至于最大化以后，旁边就有好多的空白。请问这种问题怎么解 决，怎么可以在使用 Qt desinger 的情况下，也让控件可以自动调整大小以适应窗口大小 的改变？

答：你的这个问题很典型，这是由于你没有为整个的窗体设置一个顶级布局的缘故。 设置了顶级布局之后，窗体上的所有窗口部件都处于它的管理之下，就能够实现整体的联动 了。

要查看你的窗体有没有设置顶级布局，可以在对象查看器中可以看到是否设置了布 局，以及布局是什么类型的。

另外，如果你对布局与窗体边界间的空白不是很满意，可以设置它，就是设置 margin，你可以阅读本章的内容，在里面有详细的使用方法。

问：容器内部的部件如何布局 我有三个容器已经布局好了,现在想将容器内部部件进行布局,使之能够跟所属容器同时变大或缩小,这个该怎么办呢？

答：添加布局的方法都大致如此：

第 1 步，定义布局内要用到的窗口部件

第 2 步，定义一个布局

第 3 步，把第 1 步定义的窗口部件加入到布局中

第 4 步，把布局安装到它的父窗口上

其中第 4 步就是针对你说的问题，你只需要把那些容器的实例作为布局的父窗口就可 以了。

11.9 总结与提高

布局管理是 Qt 程序开发最基本的技能之一，可以说，只要是使用 Qt 开发应用程序， 就会用到布局管理。在本章中，依次讲解了 Qt 中布局的基本概念和分类，基本布局的创建 方法和步骤，复杂的布局如栈布局、分裂器布局等的使用方法，最后对布局管理的经验进行 了总结。

设置布局管理有两种常见的做法，手写代码方法和使用 Qt Designer 设置布局的方 法。建议读者优先掌握手写代码的方法，在使用 Qt Designer 设置布局时，建议与 Qt Creator 结合使用，效果更佳。使用布局管理确实有着许多优势，但它也不是万能的，一般 在大型的应用程序开发中，通常是将布局管理与各种摆放窗口部件的方法结合起来使用。

第 12 章 使用 Qt Creator

本章重点

- 了解 Qt Creator 支持的平台和版本情况
- 了解 Qt Creator 的组成和主要特点
- 掌握 Qt Creator 的几种不同模式和操作方法
- 掌握 Qt Creator 各个组成部分的操作方法
- 掌握使用 Qt Creator 开发应用程序的流程和基本步骤

12.1 Qt Creator 概览

Qt Creator 是 Nokia 出品的 Qt4“官方”的跨平台 IDE，它能够在 Linux、Mac OS X 以 及 Windows 等绝大多数平台上使用，它的界面简洁大方、操作便捷顺畅，是广大 Qt 开发人员的首选 IDE 之一。

我以写书时最新的 Qt Creator1.2.1 版为例，向大家详细介绍它的使用方法。 当你安装了 Qt SDK 后，Qt Creator 就已经安装到了你的系统中了。你也可以单独安 装 Qt Creator，但是我并不推荐这种做法，因为你在开发时仍然需要 Qt SDK 中的其它内容。有

关 Qt Creator 的安装这部分内容，请参看第 4 章。

12.1.1 支持的平台

Qt Creator 支持以下平台或更高的平台版本。

- Windows XP Service Pack 2
- Windows Vista
- (K)Ubuntu Linux 5.04
- (K)Ubuntu Linux 7.04 32 位和 64 位版本
- Mac OS 10.4 及更高版本

小贴士：如果在以上平台采用源代码编译的方式安 装 Qt Creator 的话，需要使用 Qt 4.5.0 或更高的版本。笔者也建议读者朋友尽量使用 Qt 4.5.0 及以上的版本。

12.1.2 主要特点

Qt Creator 包含有如下重要特性：

1. 高度智能的代码编辑器 支持代码高亮以及自动完成功能。

2. Qt 4 工程向导（Project Wizard）

使用 Project Wizard，用户可以轻松创建基于控制台的应用程序 、GUI 应用程序以及 C++ 类库等多种类型的工程。

3. 集成帮助功能

在 Qt Creator 中可以查阅相关的 Qt 文档和示例程序。

4. 集成 Qt Designer 功能

无缝集成了 Qt Designer，使用者不用单独打开 Qt Designer 即可完成用户界面的创建 工作。用户只需在 Project Explorer 中双击.ui 文件，即可调用集成的 Qt Designer 完成编 辑工作。

5. 模块间智能导航功能

用户可以通过使用快捷键，来准确定位文件信息以及在不同的模块间导航。

6. qmake 工程文件格式化功能

支持将.pro 文件作为工程描述文件。

7. 集成调试器

可以使用 GNU 的 GDB（开源版）以及 Microsoft 的 CDB 作为调试器（商业版）。

12.2 Qt Creator 的组成

Qt Creator 主要由菜单（Menu Bar）、模式选择器（Mode Selectors）、项目浏览器（Project Inspector）、代码编辑器（Code Editor）、输出面板（Output Panes）、边栏（Sidebars）、 快速导航面板（Quick Open Pane）等组件构成。

在图 12-1 中显示了 Edit 模式下，Qt Creator 主要的组成部件以及布局情况。其它模式 下的组成和布局我们将结合模式选择器（ Mode Selectors）讲解。

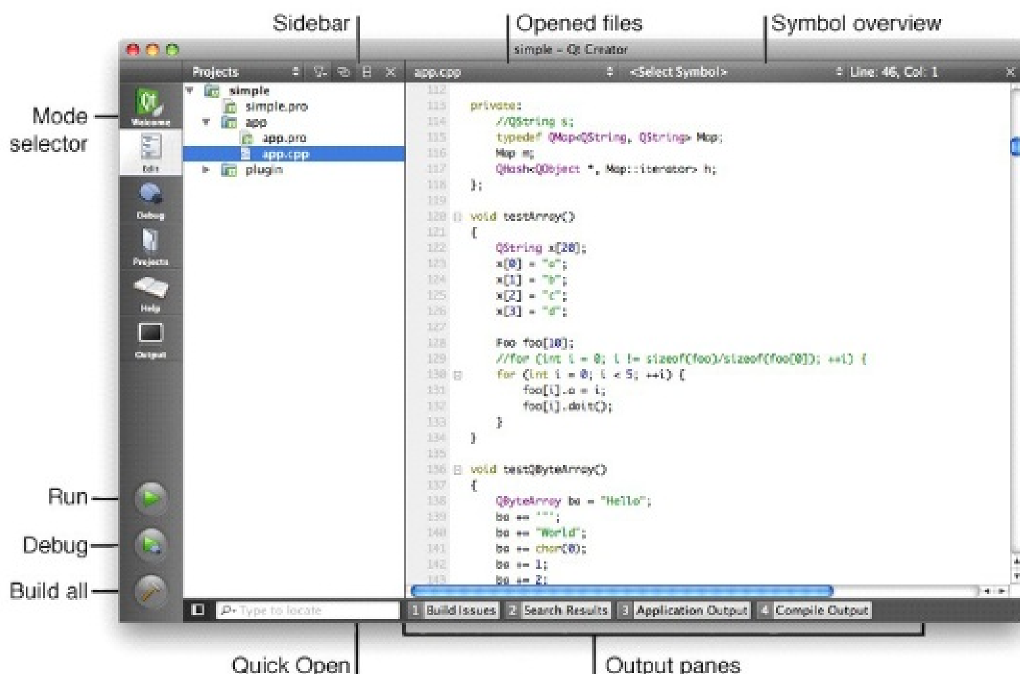


图 12-1 Qt Creator 布局架构

12.2.1 模式选择器（Mode Selectors）

Qt Creator 有 6 种工作模式可供开发者选择，分别是: Welcome, Edit, Debug, Projects, Help, 和 Output。

模式选择器允许开发者在处理不同的任务时可以快速的切换工作模式，比如编辑代码 、 浏览帮助、设置编译器环境等。在切换时，你可以通过在界面左边的模式选择器分栏上单击 鼠标左键，或者使用相对应的快捷键。当你使用特定模式下才有的动作时，也会使你自动切 换到相应的模式，比如当你依次单击菜单 Debug/Start Debugging 时，Qt Creator 将自动切 换到 Debug 模式下。

1.欢迎模式（ Welcome Mode ）

如图 12-2 所示，在该模式下 Qt Creator 显示一个欢迎屏幕。在这个模式下，你可以快 速的载入最近的人机对话或者是独立的项目 ，也可以向 Qt Creator 项目组提供反馈意见，甚 至加入到 Qt Creator 项目组中，成为其中的一员。

这个屏幕分为 3 个专栏：Getting Started、Develop 和 Community。在 Getting Started 专栏下，你可以学习 Qt Creator 的使用以及 Qt4 编程的相关知识和技能；在 Develop 专栏 下，你可以快速的恢复与 Qt Creator 的上一次对话过程，也可以打开新近使用的项目或者创 建一个新的项目；在 Community 专栏下，你可以获取 Qt Labs 网站上的新闻，也可以访问流 行的 Qt 站点。

当你在命令行下面调用 Qt Creator 时，在不附加额外的参数的情况下将进入到这个欢 迎模式下。

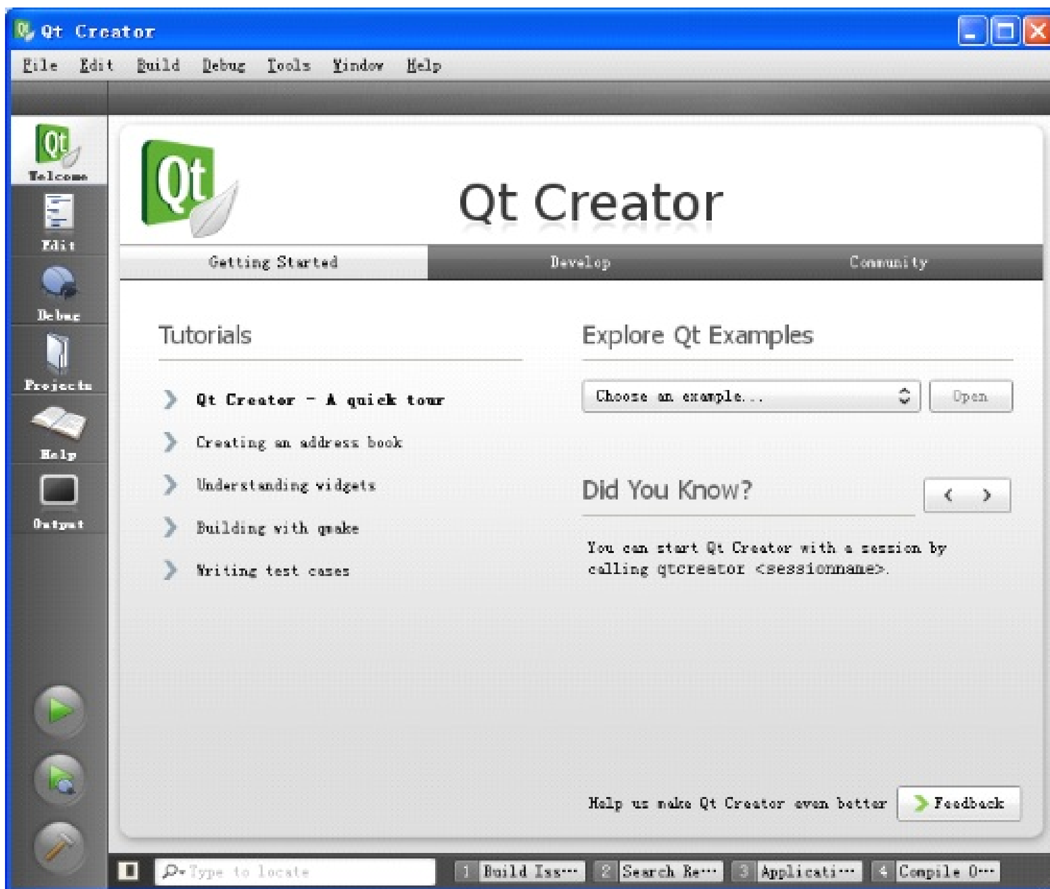


图 12-2 欢迎模式界面

2.编辑模式（ Edit Mode ）

如图 12-3 所示，在 Edit 模式下，你可以编辑项目和源代码文件，在模式选择器右边一 点的边栏（sidebar）上点击，你就可以在不同的文件中导航了。

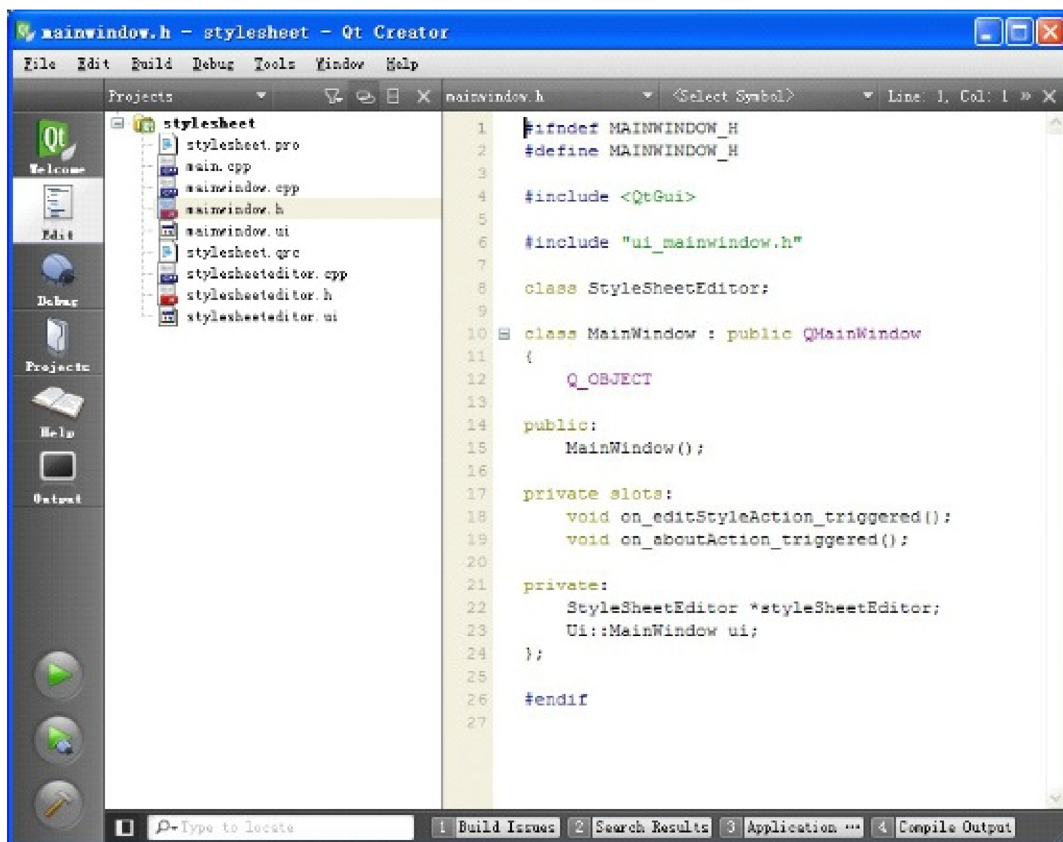


图 12-3 编辑模式界面

3.调试模式（ Debug Mode ）

如图 12-4 所示，Qt Creator 提供了多种不同的方式辅助程序员查看应用程序运行的状 态来调试程序。后面我们会结合具体例子讲解。

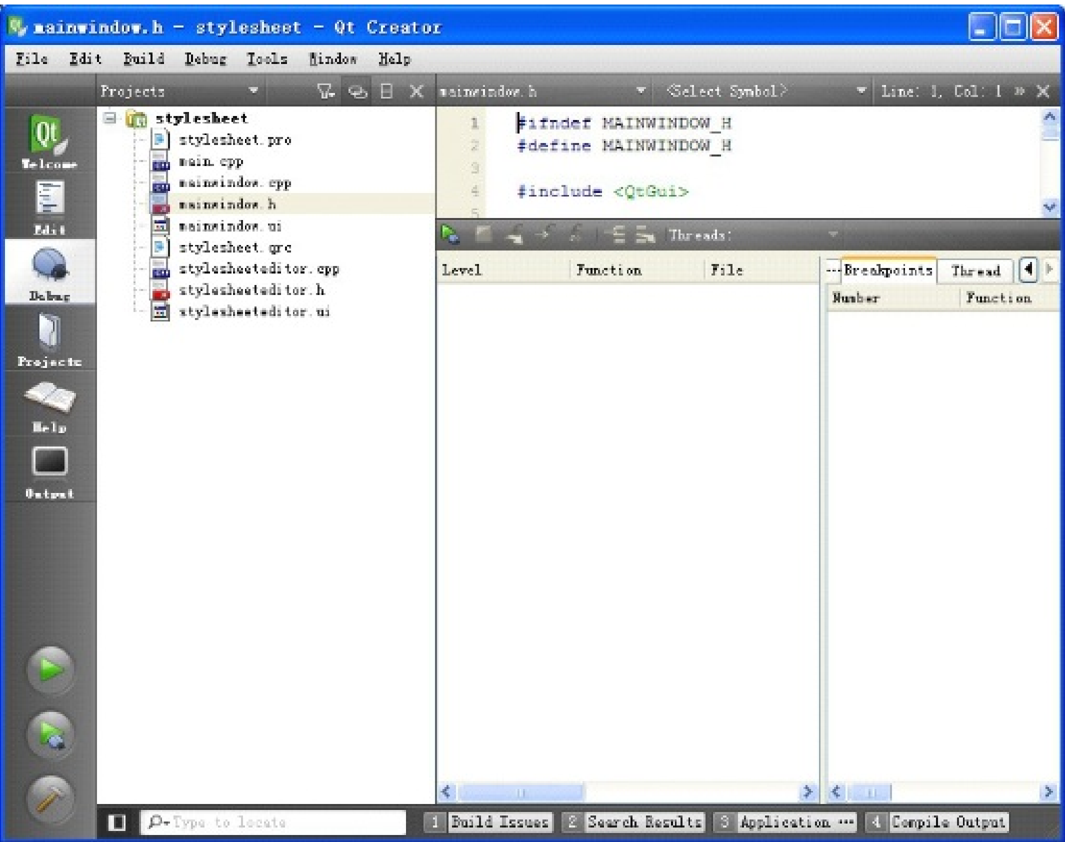


图 12-4 调试模式界面

4.项目模式（ Projects Mode ）

如图 12-5 所示，在项目模式下，首先你可以查看所有项目的列表，并可设置以哪一个 项目为当前的活动项目。然后可以选定项目，针对构建（build），运行（run）以及代码编辑 器等多个方面进行详细设置。

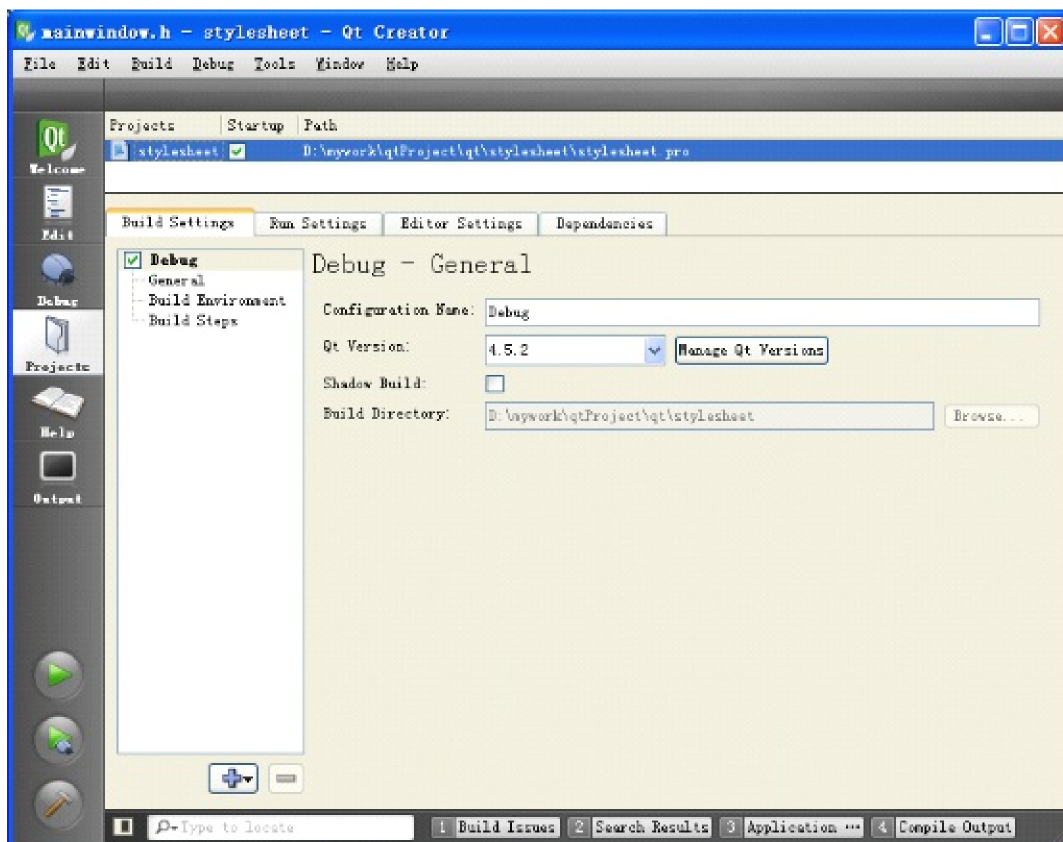


图 12-5 项目模式界面

5.帮助模式（ Help Mode）

如图 12-6 所示，主要是无缝集成了 Qt 的文档和示例中的相关内容，你可以不必另行打 开 Qt Assistant，就可以在 Qt Creator 的 Help 模式下获得帮助。

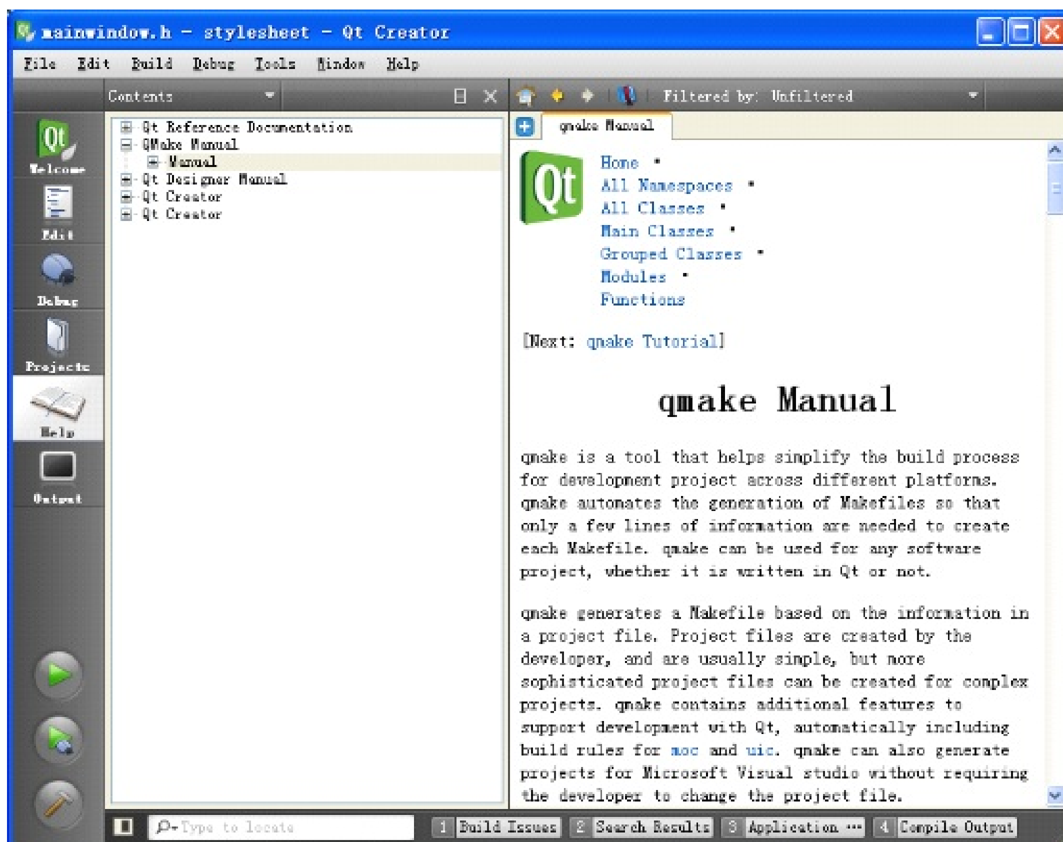


图 12-6 帮助模式界面

6.输出模式（Output Mode）

如图 12-7 所示，你可以在 Output 模式下，观察各种流程的细节，比如 qmake 以及应 用程序的编译、构建情况。这些信息你也可以再输出面板里面获得（ Output Panes）。

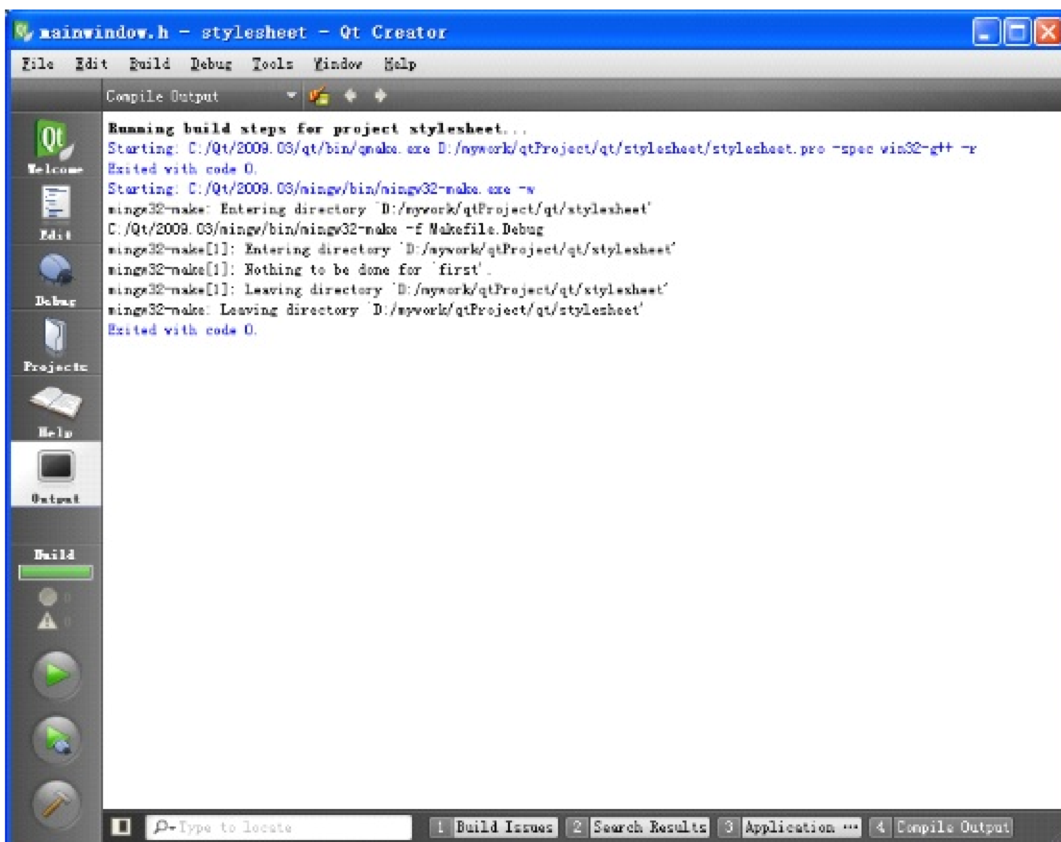


图 12-7 输出模式界面

12.2.2 输出面板（Output Panes）

Qt Creator 的输出面板主要由 4 个子面板组成，分别是： Build Issues, Search Results,Application Output,和 Compile Output。它们在所有的模式下均可以使用。

1.构建过程和结果（Build Issues）子面板

如图 12-8 所示，该面板主要显示与构建相关的信息，例如警告信息、错误信息等等， 并且指出了该产生该信息的具体位置以及可能的原因。

图 12-8 构建的流程与结果（Build Issues）

2.搜索结果（Search Results）子面板 该面板提供了执行搜索动作后的结果输出显示， 搜索的范围可以是全局的，也可以是具体局部的，比如你可以在某一个指定的文档中搜索某个词组，也可以把范围扩大到所有项目或者是电脑上的硬盘目录等等。举例来说，我们在 TextFinder 目录下面搜索含有 “TextFinder”这个词，如图 12-9 所示即是搜索的结果显示。

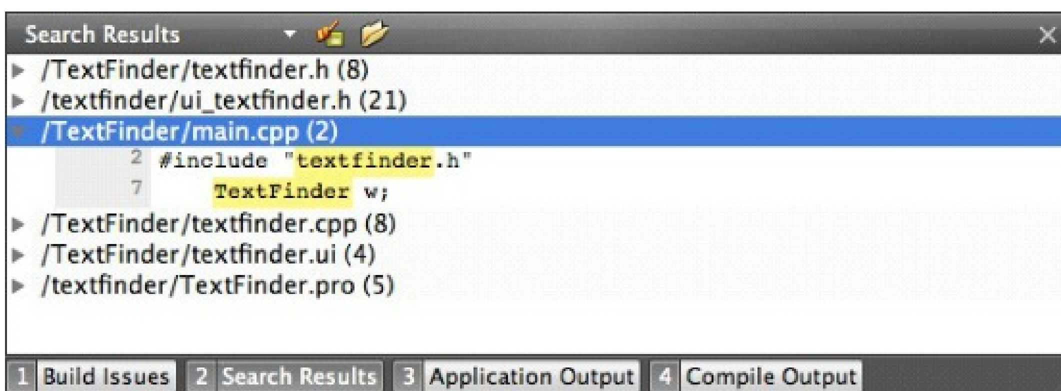


图 12-9 搜索结果（Search Results）

3. 应用程序输出子面板

如图 12-10 所示，应用程序输出子面板显示了应用程序的运行状态，包括正常运行以及 Debug 模式下的信息，比如你可以在程序中调用 qDebug()函数来查看输出情况。

图 12-10 应用程序（构建结果）的输出（Application Output）

4. 编译（Compile）子面板

如图 12-11 所示，编译子面板显示了所有来自编译器的输出信息，实际上它包含了更为详细的输出信息，包括 Build Issues 子面板显示的信息。

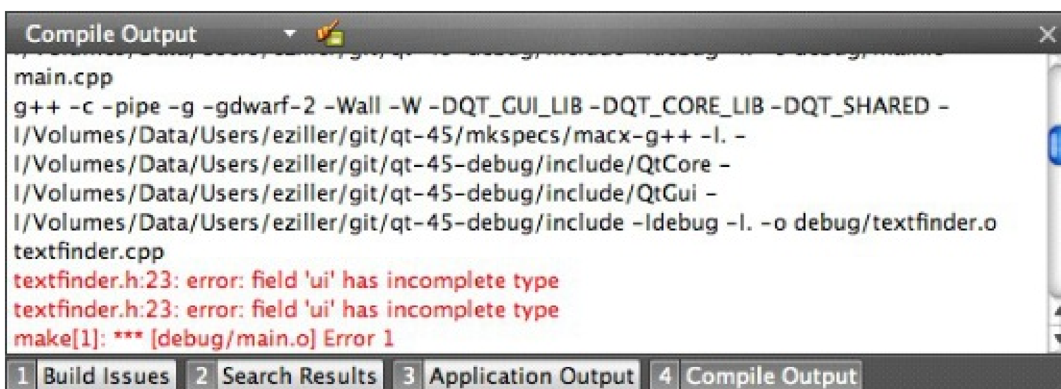


图 12-11 编译情况的输出（Compile Output）

12.2.3 代码编辑器（Code Editor）

代码编辑器辅助开发者创建、编辑代码，并可在其间导航。它具有代码高亮、代码自动完成、上下文提示以及内嵌代码错误指示等特性。

1. 属性设置

可以依次点击【Tools】→【Options...】→【Text Editors】，来设置代码编辑器的各 种属性。

图 12-12 显示了如何设置 Font&Colors（字体和颜色）属性。

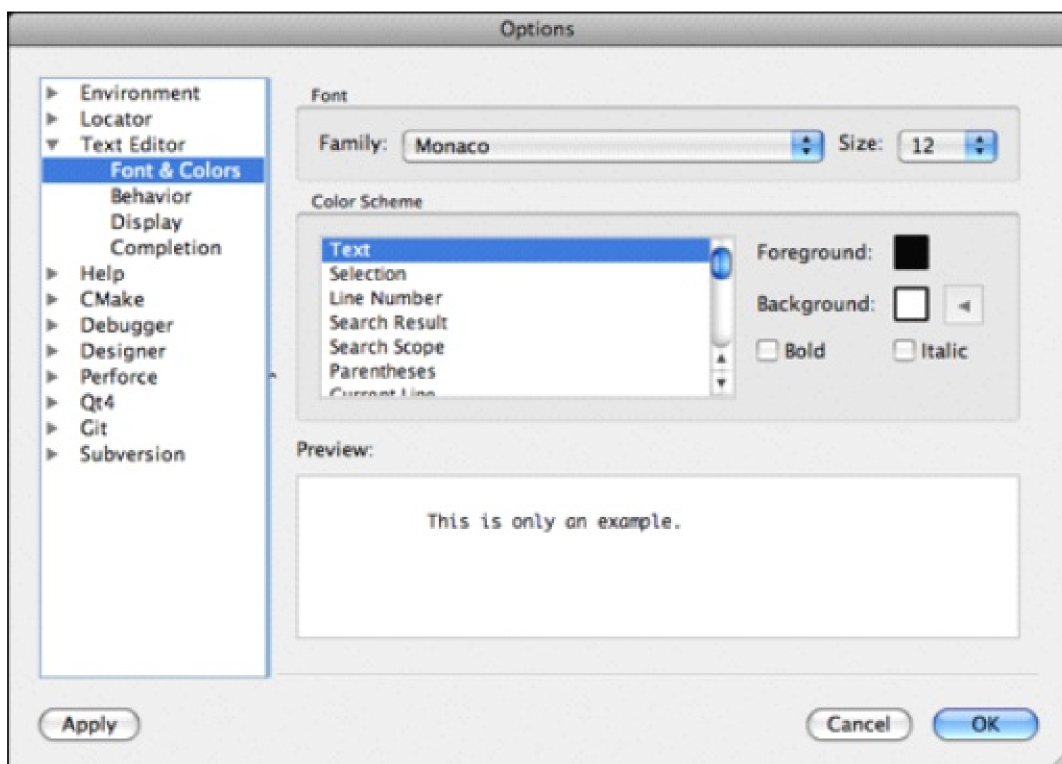


图 12-12 设置代码编辑器的字体颜色（Font&Colors）属性

图 12-13 显示了如何设置 Behavior（行为）属性。

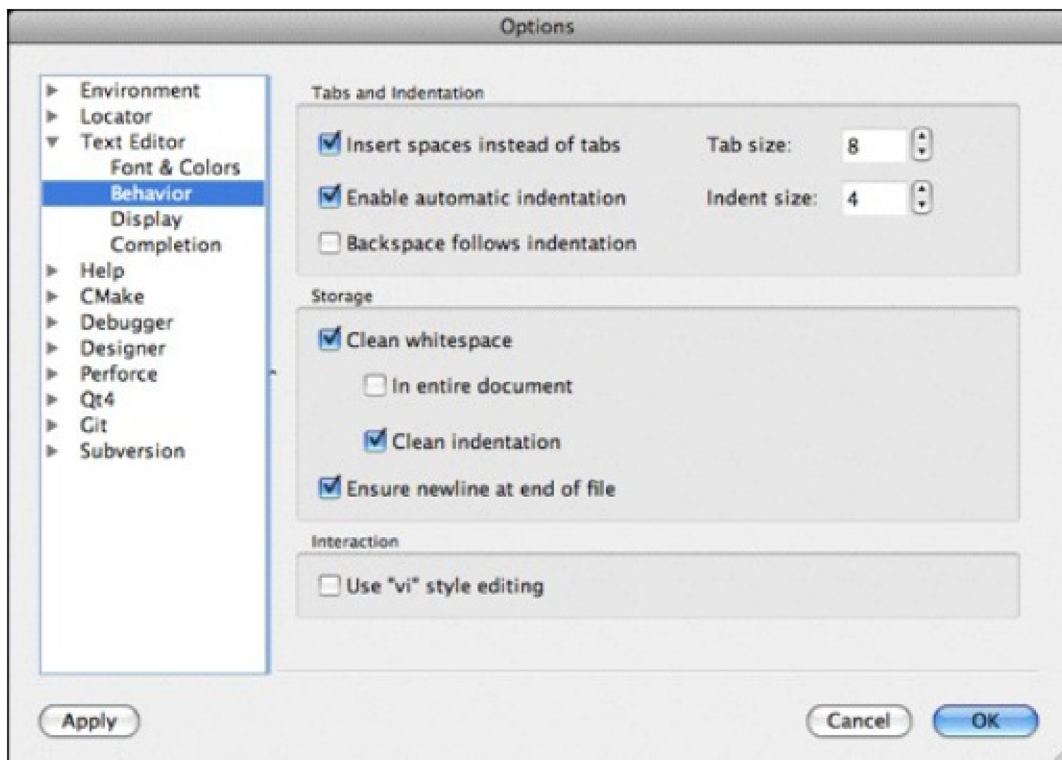


图 12-13 设置代码编辑器的行为（Behavior）属性

图 12-14 显示了如何设置 DisPlay（显示）属性。

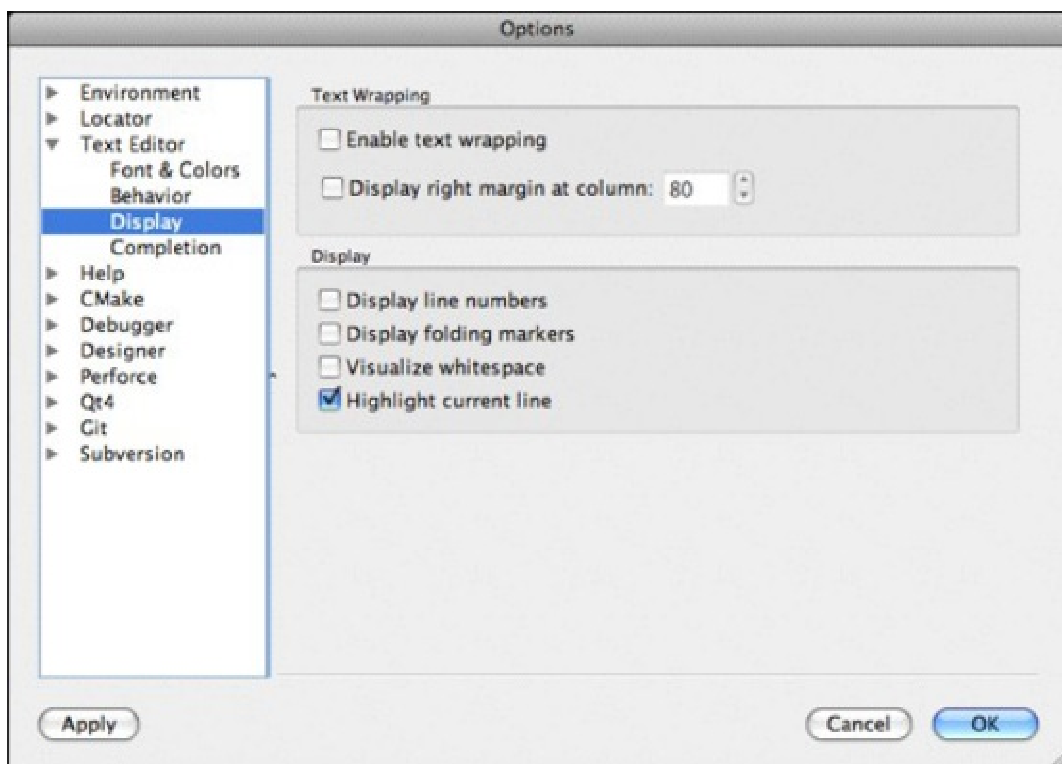


图 12-14 设置代码编辑器的展现（Display）等属性

图 12-15 显示了如何设置 Completion（代码完成）属性。

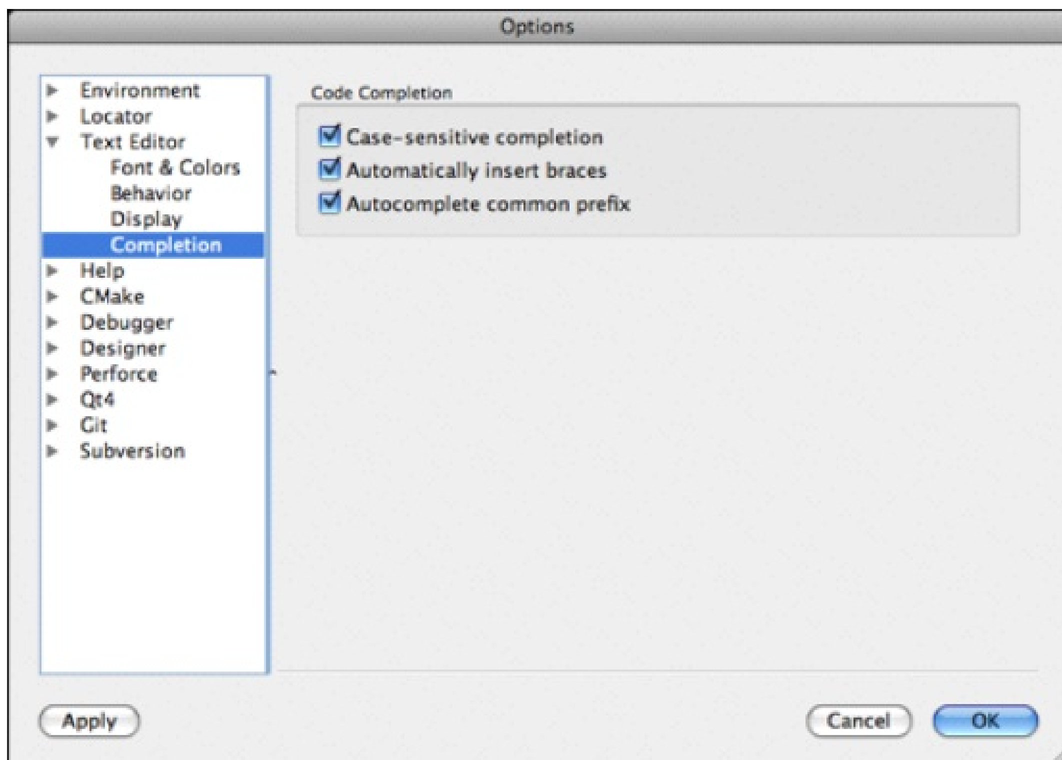


图 12-15 设置代码编辑器的自动完成（Completetion）等属性

2. 快捷键

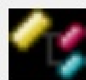
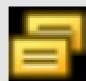
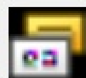
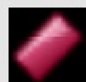
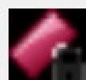
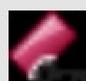
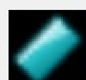
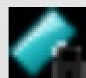
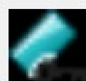
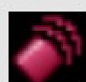
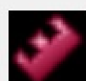
Qt Creator 的代码编辑器支持很多的快捷键，表列出了常用的一部分：

表 12-1 代码编辑器支持的快捷键

- 代码块间导航 Ctrl+[和 Ctrl+]，一般我们常用在比如在{}代码块间导航
- 选中代码块/取消选中代码块/选中上级代码块 Ctrl+U / Ctrl+Shift+U / 再次按下 Ctrl+U
- 向上/向下移动某行代码 Ctrl+Shift+Up / Ctrl+Shift+Down
- 代码自动完成 Ctrl+Space
- 格式化缩进 Ctrl+I
- 代码块折叠/展开 Ctrl+< / Ctrl+>
- 声明注释或取消注释 Ctrl+/
- 删除一行代码 Shift+Del
- 在类的头文件和实现文件间切换 F4
- 增大或缩小字体的大小 Ctrl键+鼠标滚轮
- 在声明和定义之间转换 F2 和 Shift+F2 键适用于名字空间、类、方法、变量、宏等.
- 切换到外部的编辑器 依次点击菜单 Edit -> Advanced-> Open in external editor

3. 代码完成功能（ Code Completion ）

当你在代码编辑器中输入某个词组时，系统会自动弹出一个上下文提示窗口， 里面列举了可能符合你的意图的完整代码，这个上下文提示窗口又被称为是 “代码完成提示盒子”，其中常见的类别有类、名字空间、方法、变量、宏以及关键字等。表 12-2 显示了这些常见类别 以及所对应的图标。

	类（A class）
	枚举常量（An enum）
	枚举常量值（An enumerator (value of an enum)）
	方法（A function）
	私有方法（A private function）
	保护方法（A protected function）
	变量（A variable）
	私有变量（A private variable）
	保护变量（A protected variable）
	信号（A signal）
	槽（A slot）

私有槽（A private slot）
保护槽（A protected slot）
关键字（A keyword）
宏（A macro）
名字空间（A namespace）

表 12-2 常见类别图标

12.2.4 会话管理器（Session Management）

在 Qt Creator 中，一个会话（session）指的是用户与 Qt Creator 交互的一次过程， 可以包括加载的项目、打开的文件以及代码编辑器的设置等等 。当你运行 Qt Creator 时，你 已经开启了一个的对话，Qt Creator 会将它记录下来。如图 12-16 所示，你可以依次点击【File】→【Session】→【Session Manager...】来创建和管理对话。

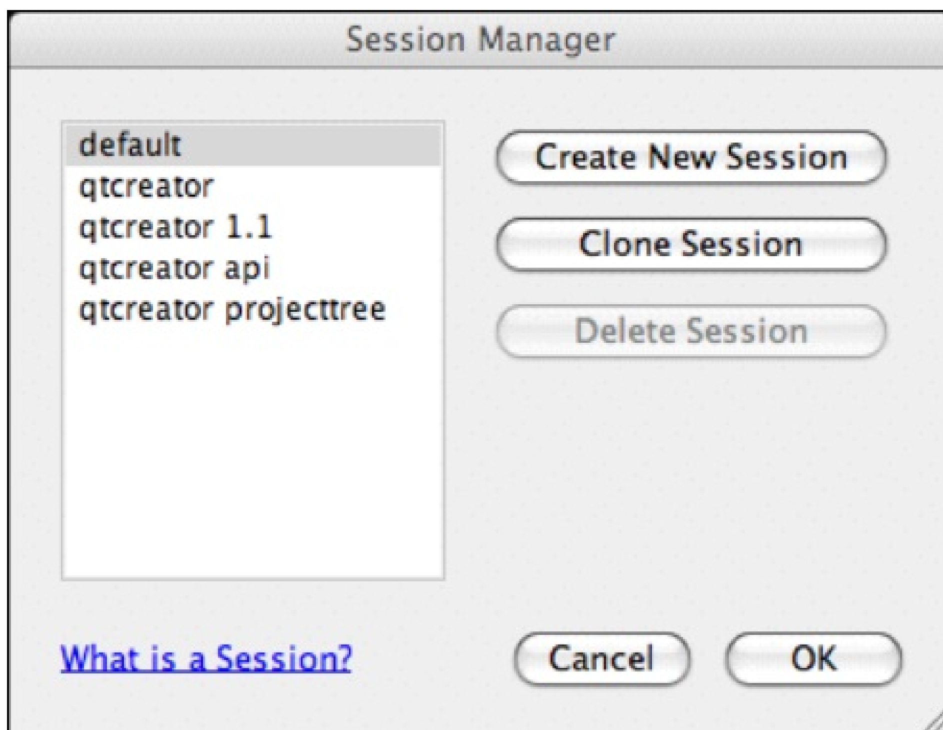


图 12-16 会话管理器

要在不同的对话间切换，你可以依次点击【File】→【Session】来切换实现，如图 12-17 所示。如果你没有创建新的对话，并且没有选择任何对话，那么 Qt Creator 将一直使用默认 的对话。

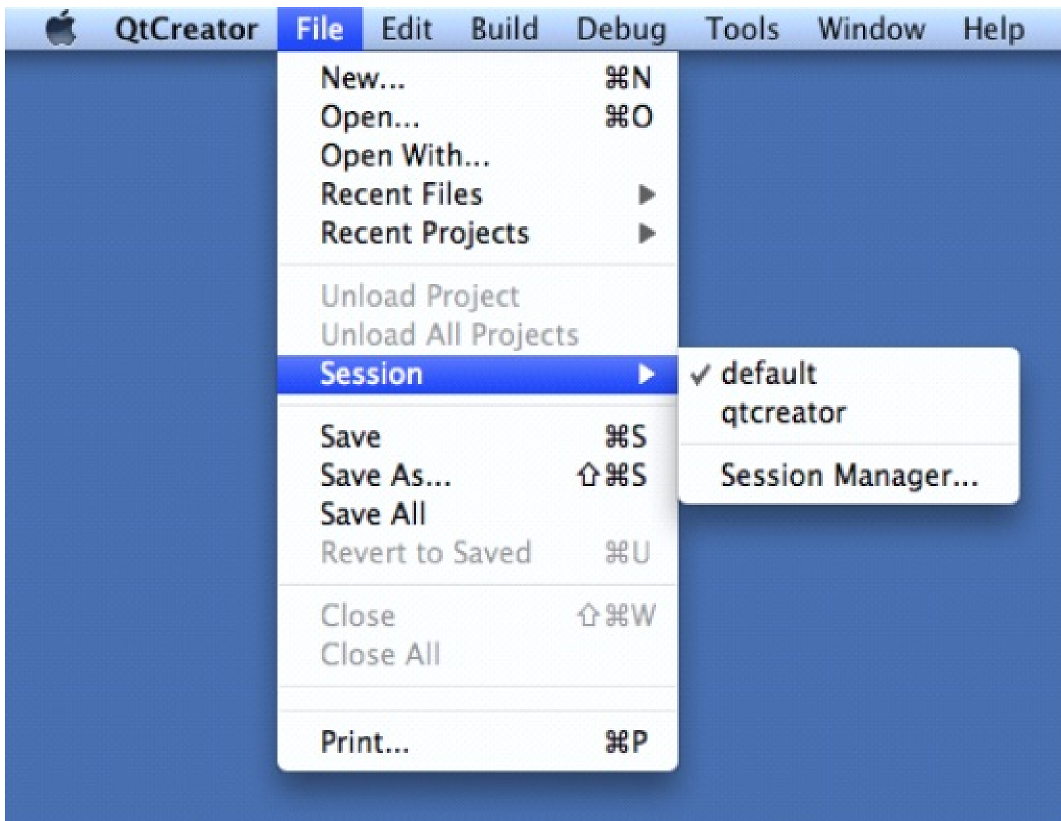


图 12-17 切换会话

12.2.5 Qt 帮助集成功能（ Qt Help Integration ）

在 Qt Creator 中使用帮助，有两种主要的方式，一种是随时按下 F1 键，一种是切换到 Help 模式下，Qt Creator 使用插件的方式将 Qt 的文档和示例集成进来。图 12-18 示例了使 用 F1 键的方式，你可以选中某个词或者类名，甚至整条句子等，然后按下 F1 键，在 Qt Creator 的右边将增加一个面板，在里面显示了文档中有关条款的内容。

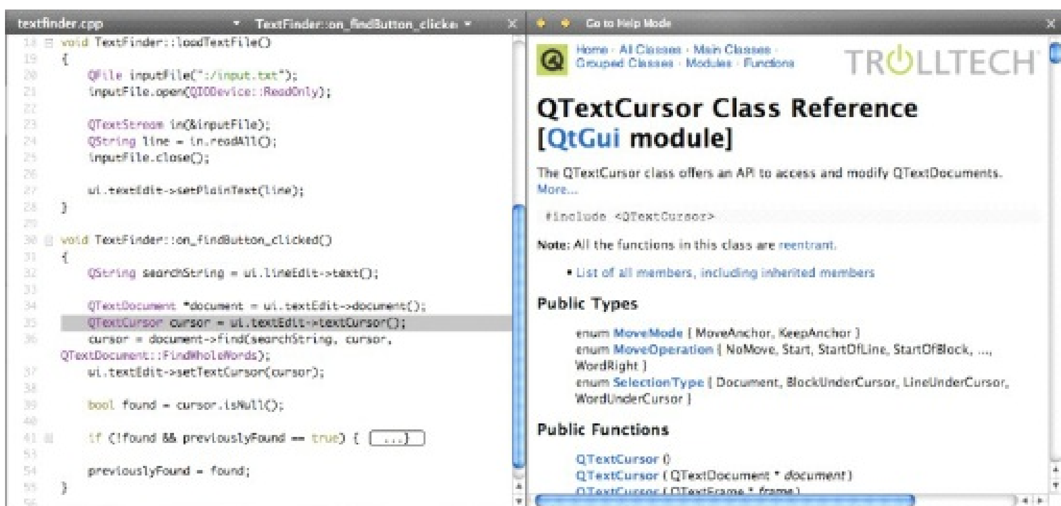


图 12-18 查阅帮助

12.2.6 Qt 设计师集成功能（ Qt Designer Integration ）

如图 12-19 所示，在使用 Qt Creator 开发应用程序时，常见的用法是用鼠标左键双击.ui文件，即可打开 Qt Creator 的 Qt Designer 集成功能。你可以看到，Qt Creator 已经与 Qt Designer 完全集成在一起了。这样你就可以在不单独运行 Qt Designer 时，在 Qt Creator中完成应用程序界面的设计，并且与 Qt Creator 的项目管理以及其它功能在一起获得对 Qt 项目的完整把握。

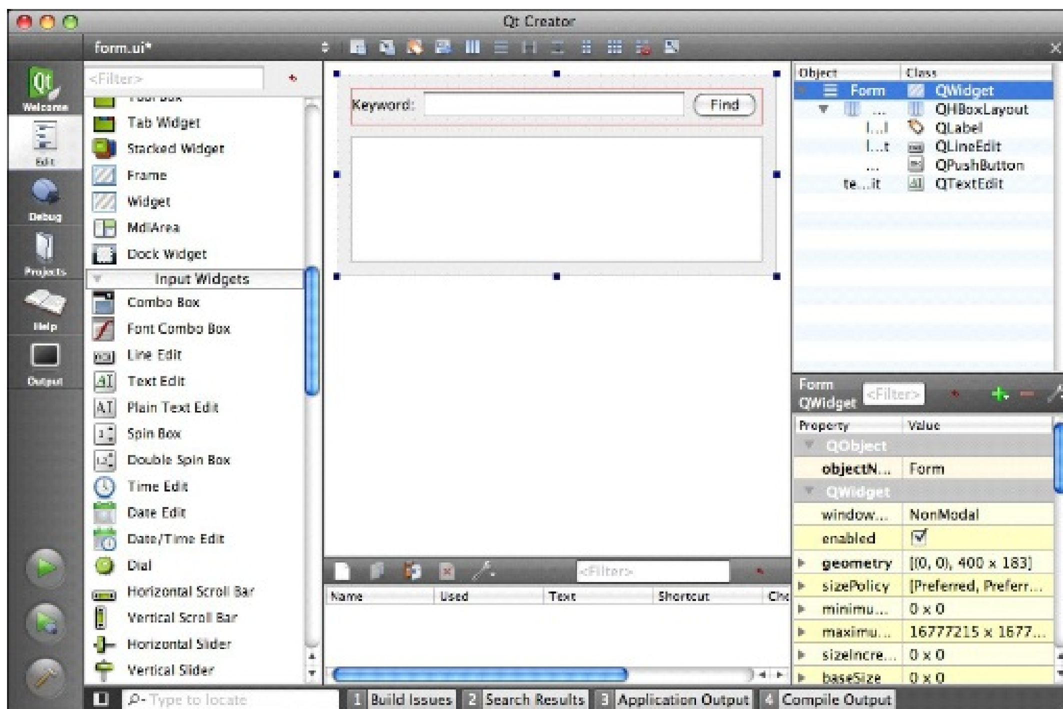


图 12-19 在 Qt Creator 中集成 Qt Designer

12.3 快捷键和常用技巧

众所周知，有的开发者喜欢使用鼠标完成常用的操作， 而有些开发者更喜欢使用键盘快捷键。Qt Creator 的操作简单明了， 它为主要的功能提供了对应的快捷键组合以及导航， 这将帮助开发者提高开发效率。

Qt Creator 提供了丰富的快捷键来辅助开发者加快开发进程 。下表中列出了常用的快捷键。

表 11-3 快捷键和常用技巧

功能	快捷键
激活 Welcome 模式（Activate Welcome mode）	Ctrl + 1
激活 Edit 模式（Activate Edit mode）	Ctrl + 2
激活 Debug 模式（Activate Debug mode）	Ctrl + 3
激活 Projects 模式（Activate Projects mode）	Ctrl + 4
激活 Help 模式（Activate Help mode）	Ctrl + 5
激活 Output 模式（Activate Output mode）	Ctrl + 6
查找（Find）	Ctrl + F
查找下一个（Find next）	F3
回到代码编辑器（Go back to the code editor）	Esc
跳转至代码的某一行（Go to a line）	Ctrl + L
在页面间导航（Navigate between pages）	Alt + Left, Alt+ Right
启动调试（Start debugging）	F5
停止调试（Stop debugging）	Shift + F5
在代码的声明和定义之间切换（Toggle code declaration and definition）	F2
在类的头文件和实现文件之间切换（Toggle header file and source file）	F4
切换到边栏（Toggle Side Bar）	Alt + 0
切换到 Build Issues 面板（Toggle Build Issues pane）	Alt + 1
切换到 Search Results 面板（Toggle Search Results pane）	Alt + 2
切换到 Application Output 面板（Toggle Application Output pane）	Alt + 3
切换到 Compile Output 面板（Toggle Compile Output pane）	Alt + 4

小贴士：就笔者的体会，把键盘与鼠标操作结合起来使用，往往效率最高，因此熟悉主要功能的快捷键是很有必要的。但大家也不必强求自己非要掌握和使用哪种方式， 只要熟能生巧， 顺其自然就好。

12.4 Qt Creator 构建系统的设置

Qt Creator 的构建系统是建立在 qmake 和 make 基础之上的，设置 Qt Creator 的构建系统，本质上就是对 qmake 和 make 进行设置，只不过是以图形界面形式完成。

对 Qt Creator 构建系统的设置，默认情况下其实是对 qmake 的设置，只不过 Qt Creator 为我们提供了 GUI 界面，使得这些工作变得简单和生动起来，这就需要切换到 Projects 模式，方法是使用鼠标或者按下 Ctrl+4 组合键，当然前提是你已经打开了一个工程。如图 12-20 所示。

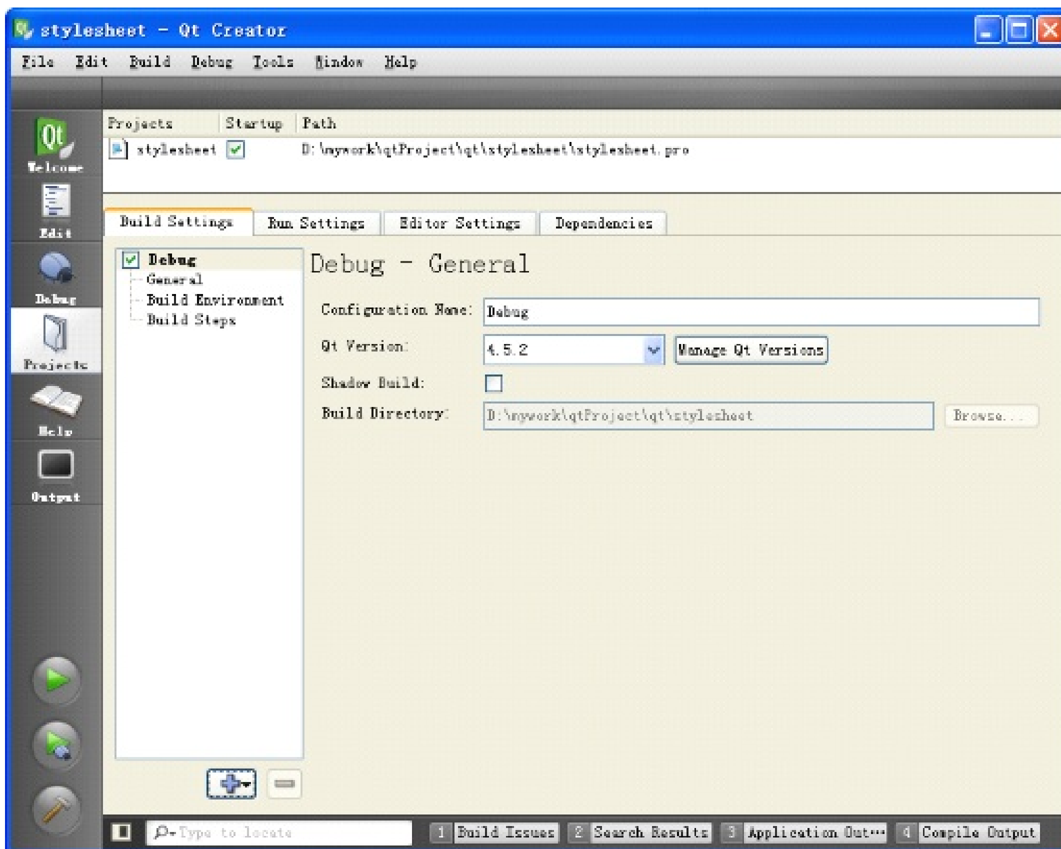


图 12-20 切换到 Projects 模式

默认情况下，Qt Creator 创建 debug 和 release 两个版本，它们都使用 Default Qt Version，每一个版本都有【General】、【Build Environment】、【Build Steps】三个分栏， 你可以在其中设置相关的内容。

在介绍如何设置之前，先了解几个常用术语。

表 11-4 常用术语

术语	含义
Auto-detected	Qt 如果你在系统的 PATH 目录中设置了 Qt 的目录，那么 qmake 将自动发现这个版本，称为 Auto-detected Qt。
Default	Qt 它默认就是 Auto-detected Qt。如果你在 PATH 中没有设置 Qt 的目录，那么 Qt Creator 将把自动寻找到的 Qt4 版本作为 Default Qt，并且你在创建新的工程时，将采用这个版本。你可以依次点击主菜单的 Tools -> Options -> Qt 4 -> Default Qt Version. 中查看 Default Qt。
Project	Qt 这是你的具体项目采用的 Qt 版本。你可以通过依次点击 Build&Run -> Build Settings -> Build Configurations 来查看并设置它。默认情况下，它等同于 Default Qt。
Shadow Build	它的机理类似于大家所熟悉的影子模式，可以命名为"以影子模式构建（项目）"采用这种模式构建时，将在 一个与你的项目的源代码目录不同的目录下进行，而你的工程源目录将是 "干净"的，不会有任何的改动。 当你的工程设置需要频繁变更时，使用影子模式以适应各种情况是最佳的选择。

在【General】标签页中，如图 12-20 所示，你可以为项目选择 Qt 的版本，要不要使用 Shadow Build 等。

在【Build Environment】标签页中，如图 12-21 所示，你可以为 Qt Creator 设置环境， 如常见的 PATH、QTDIR、LIB 变量等，当你使用 SDK 方式安装 Qt，安装程序会把这些环境为 你设置好，不必手动修改。

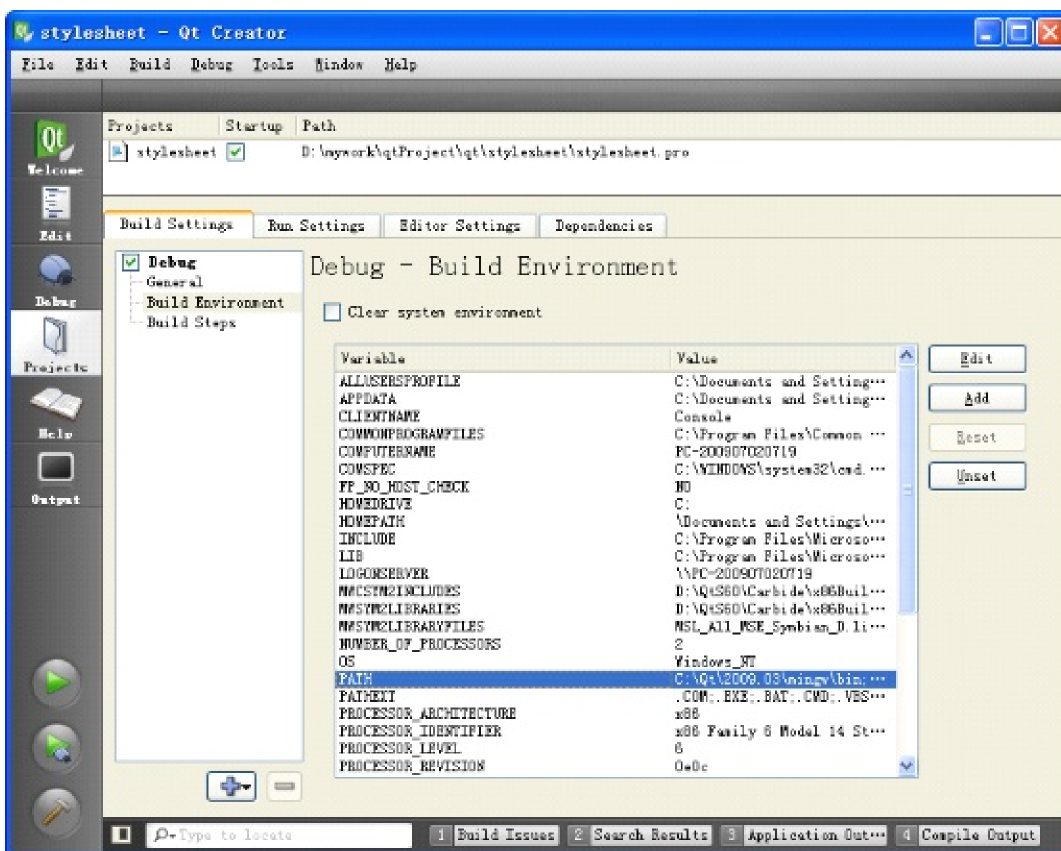


图 12-21 设置【Build Environment】标签页

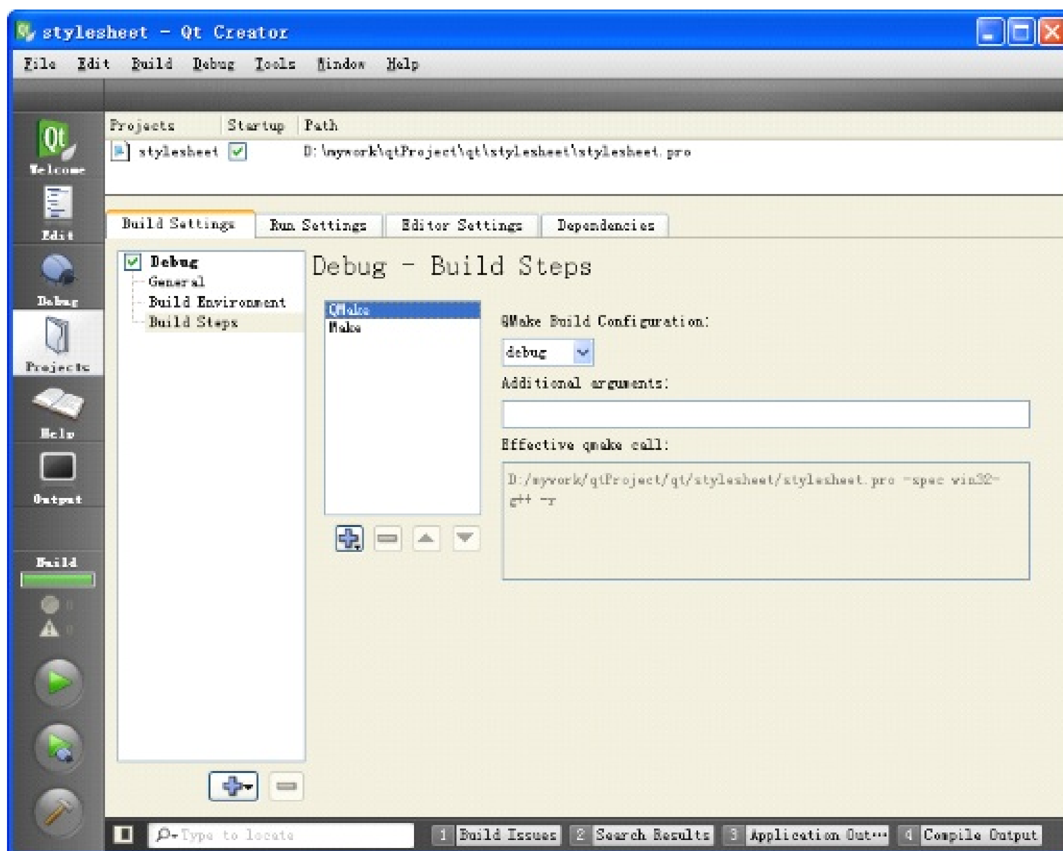


图 12-22 设置【Build Steps】标签页

在【Build Steps】标签页中，如图 12-22 所示，你可以为 Qt Creator 配置 qmake 和 make 的属性，更进一步的，你可以自定义编译的具体步骤。

12.5 处理项目间依赖关系（Dependencies）

如果你在一次对话中加载了多个项目，你就可以设置它们之间的依赖关系，这也将影响你的项目的构建顺序。具体做法是打开 Dependencies 标签页，在其中选中你的依赖，如图12-23所示，我们同时载入了两个项目：textfinder 和 stylesheet，并以 textfinder 工程为当前工程（active project），点击 Dependencies 标签页，选中 stylesheet 前面的复选框，这时 textfinder 工程就以 stylesheet 为依赖了。

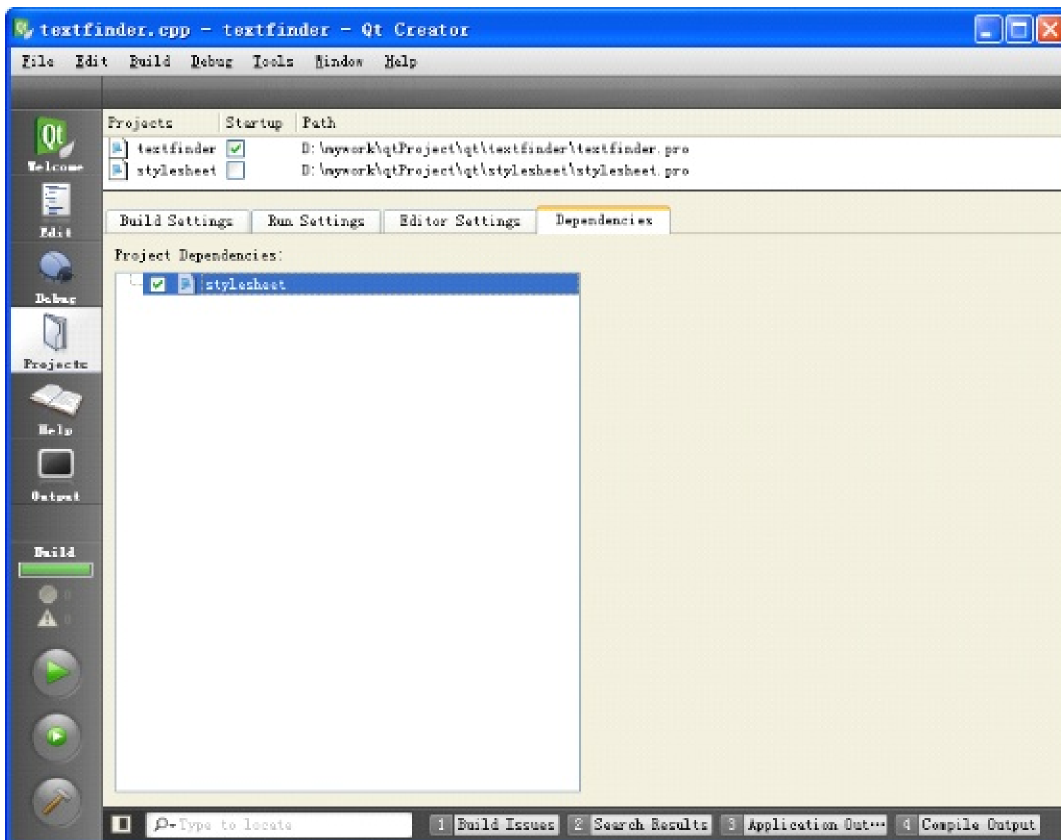


图 12-23 设置项目间的依赖关系

12.6 Qt 多版本共存时的管理

Qt Creator 允许使用多个 Qt4 版本，并且可以在不同的版本间快速切换。

当 Qt Creator 启动时，它首先会根据环境变量 PATH 中设定的目录寻找 Qt4，这个被自 动找出来的 Qt4 版本称作“Auto-detected Qt”。术语“version of Qt”指的通常也是 Auto-detected Qt。当然了，读者朋友如果只是安装了一个版本的 Qt4，并且准备只使用这 一种的话，只需要正确的设定 PATH 变量，而无需手动配置 Qt 的版本了（如果使用 SDK 方式 安装，就更为简便，Qt4 SDK 会自动设置环境变量），这一节就可以略过不看。 表 12-4 列出 了与此有关的常用术语，了解它们的含义很有必要。

此外，也可以自由的添加或删除不同的 Qt 版本。方法是依次点击菜单【Tools】→【Options...】→【Qt4】→【Qt Versions】，如图 11-24 所示，如果是在 Windows 平台上使用 MinGW 作为编译器，那么需要告诉 Qt Creator 你的 MinGW 安装到了什么地方，当然如果是使用 SDK 方式安装的 Qt，则无这一步的必要。设置它的方法是依次点击 【Tools】→【Options...】→【Qt4】→【Qt Versions】→【MinGw Directory】。

如果编译安装 Qt 时是专为 Microsoft Visual C++的，那么 Qt Creator 将自动为你设置 好这些环境变量。

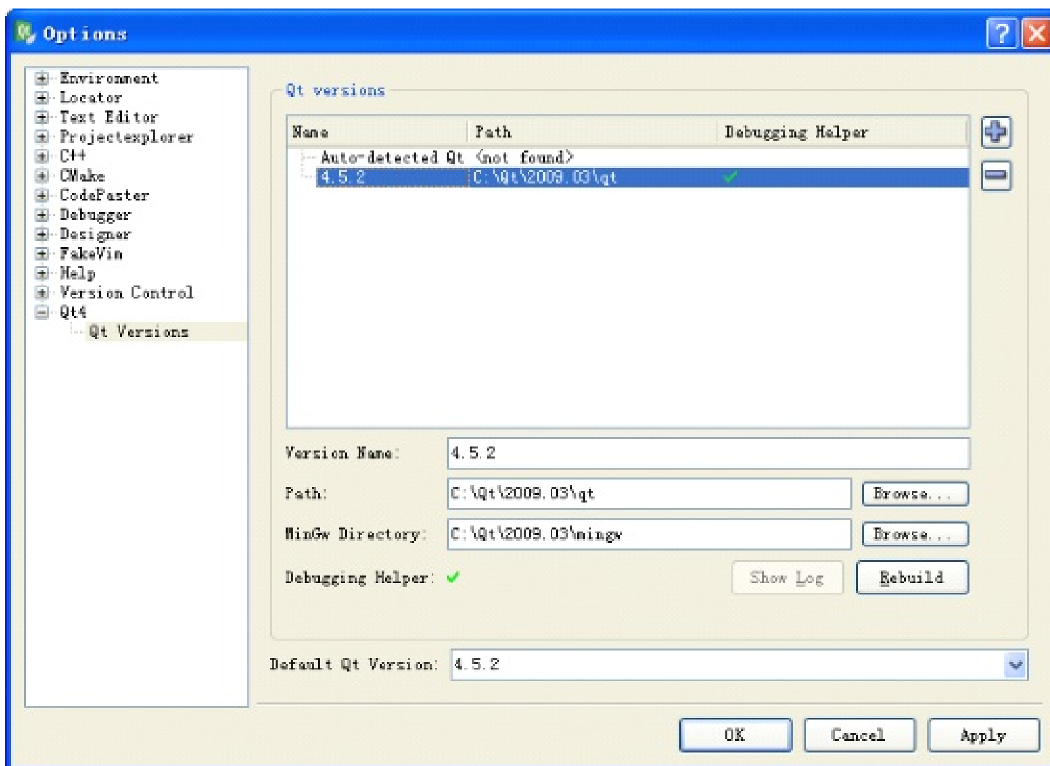


图 12-24 设置 Qt 的版本

小贴士: 上述这些设置也可以在上一节讲到的配置工程的构建系统中完成，请读者自行验 证。

12.7 使用定位器在代码间快速导航

Qt Creator 提供了一个定位器，它位于 Qt Creator 窗体的底部，它是一个智能的编辑 框（line edit）,你可以使用它在你的项目内部或硬盘上执行不同的定位（搜索）任务，如 可以定位文件、类、方法等等。与其它我们以前熟悉的 IDE 中的“搜索”器或定位器不同的 是，当你用鼠标左键在编辑框中点击时，它将弹出一个上下文窗口。如图 12-25 所示。

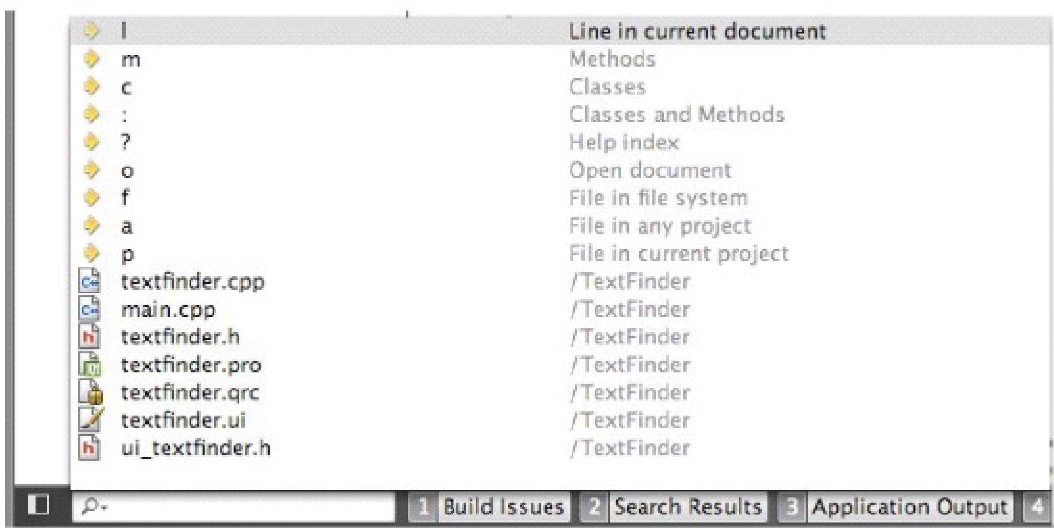


图 12-25 定位器的上下文窗口

12.7.1 如何定位文件

直接举例子吧，假设你想打开项目中 main.cpp 这个文件，那么你可以使用鼠标在定位 器上点击或者按下 Ctrl+K 键，在其中输入文件名即 main.cpp，然后按下回车键。Qt Creator 将使用代码编辑器打开它。你也可以不输入全称，而是输入关键字，或者加上 *和?这样的通 配符，这时定位器将会把所有符合条件的信息罗列出来，请你选择。在其中选择你满意的项 目，完成定位即可。

12.7.2 如何设置过滤条件

你可以为定位器设置许多不同的过滤条件 （Filters），这样你就可以执行各种不同的定位任务。下面列出了一些常用的过滤条件。

- 在你的硬盘上的任意文件（将通过文件系统查找）
- 在你定义的子目录结构中的文件
- 在你的工程文件（.pro）中提到的文件，如头文件、实现文件、资源文件、资源集 文件以及.ui 文件
- 任何打开的文档
- 在你的项目中的定义或引用的类或方法

- 在 Qt 文档中的帮助主题
- 在你的代码编辑器中的任意指定的一行代码

那么如何使用这些过滤条件呢，方法按下 Ctrl+K 键或者使用鼠标，激活定位器，然后输入冒号:，接着输入一个空格，在这个空格之后输入你的定位前缀 。举个例子，假如你需要定位 QDataStream 这个类的定义，那么就在激活定位器后，依次输入冒号、空格以及 QDataStream，如图 12-26 所示。定位器将为你列出找到的相关信息 ，选中其中的一项，按下回车键即可定位到 QDataStream 的定义，（也可使用鼠标操作，请读者自行验证）。如图 12-27所示。

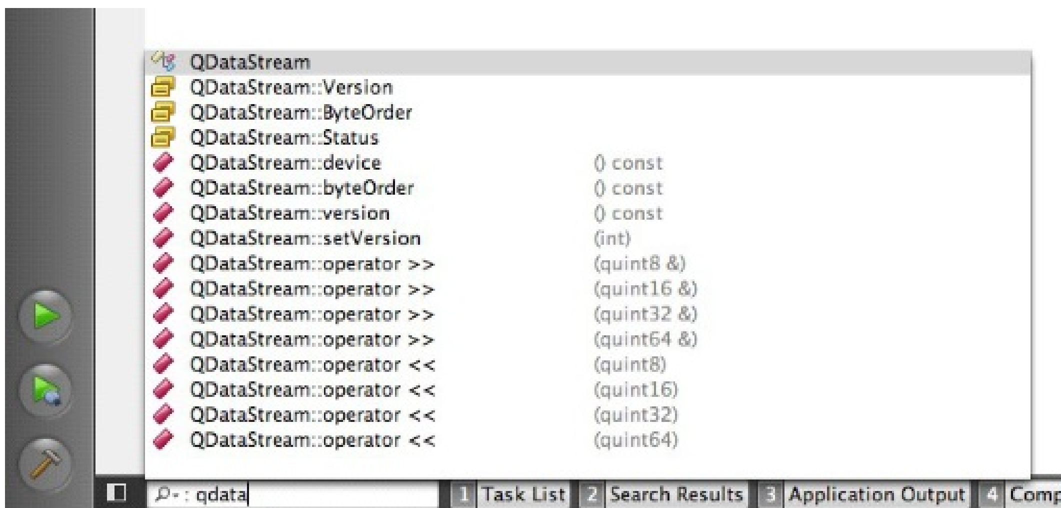


图 12-26 输入过滤条件

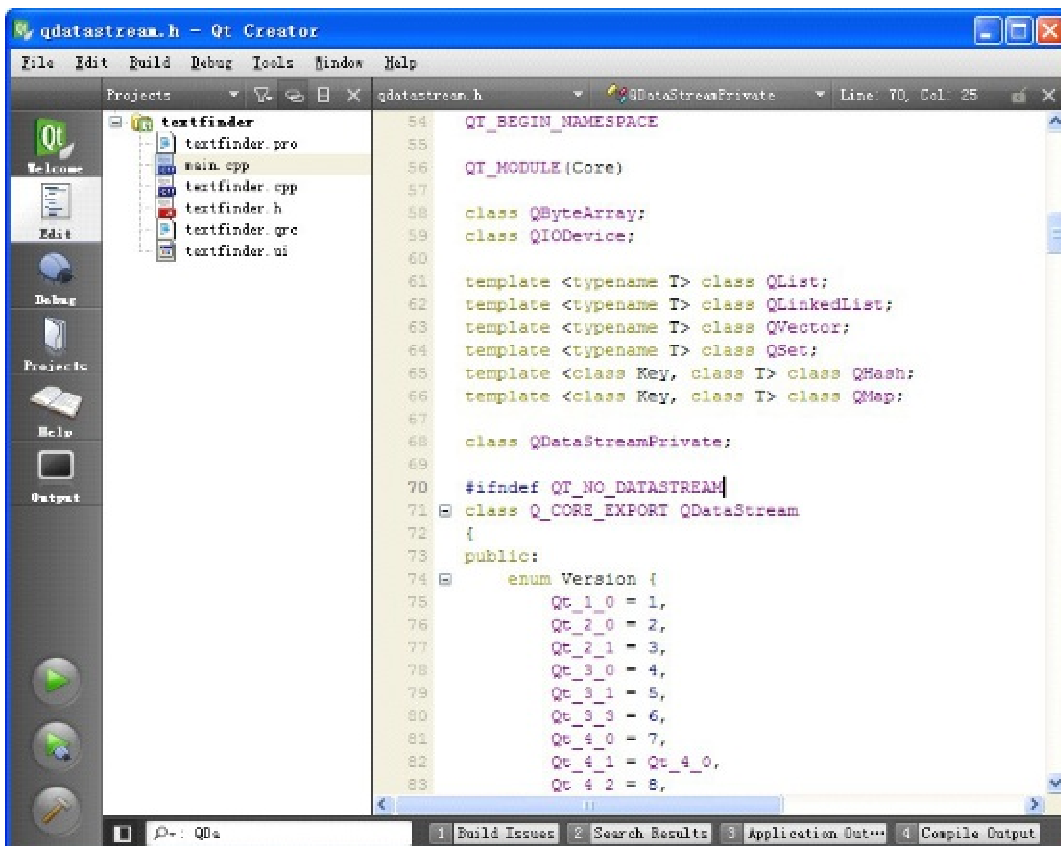


图 12-27 在代码编辑器中查看

如果你觉得系统内置的这些过滤条件不能满足你的要求，那么你可以自己定义一个 。方 法是用鼠标点击定位器上的那个用来搜索的 按钮，然后在弹出的上下文菜单上选择【Configure...】，如图 12-28 所示。

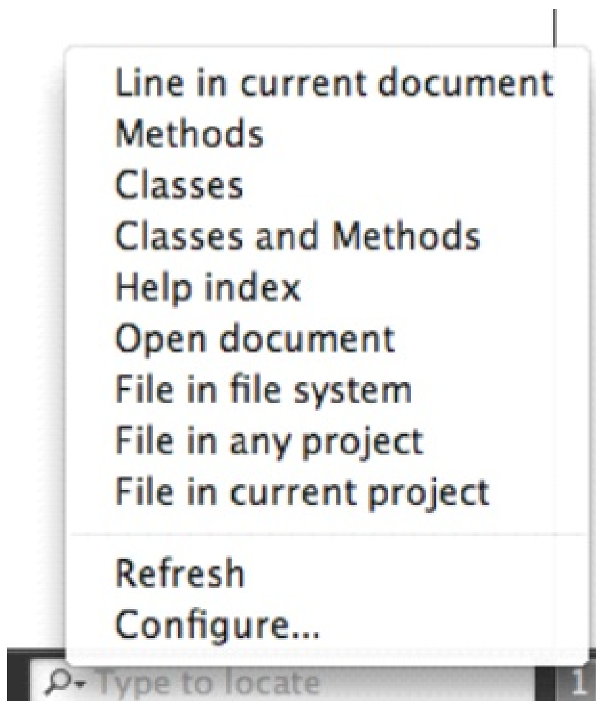


图 12-28 自定义过滤条件上下文菜单

接下来在弹出的对话框中点击【Add】按钮，创建一个新的过滤条件。这将弹出【Filter Configuration】对话框，如图 12-29 所示，为你的过滤条件起一个名字，选择搜索目录，设置可能的文件后缀名，最后再设置关键字的前缀。

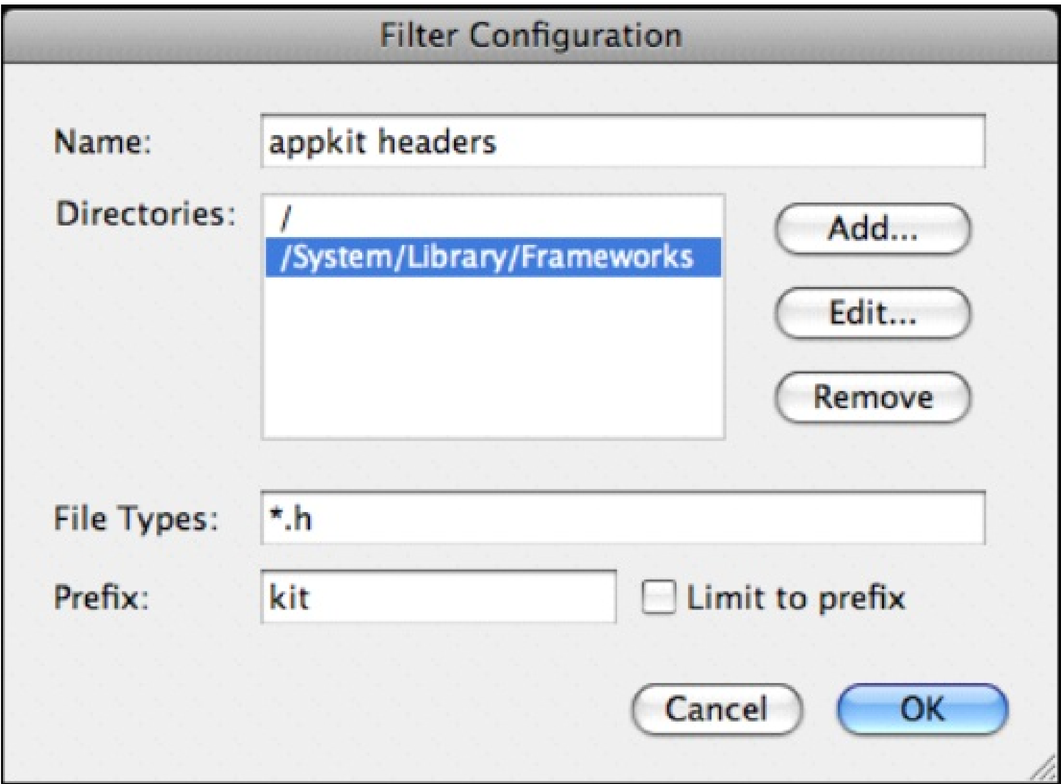
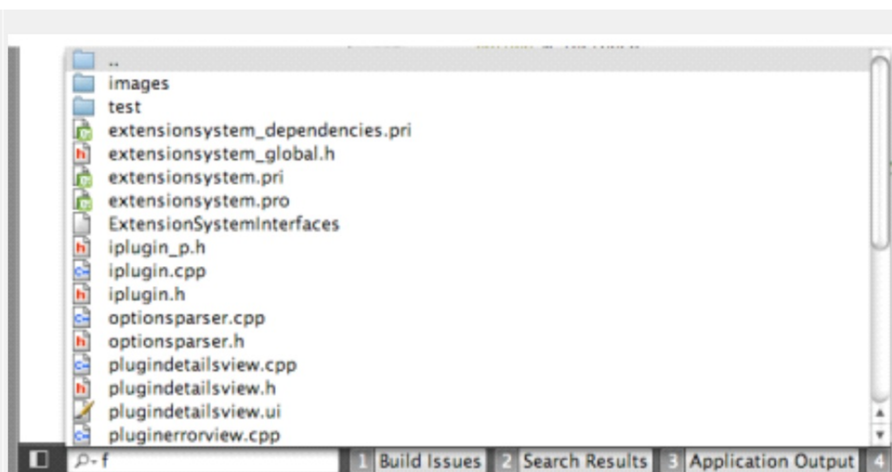
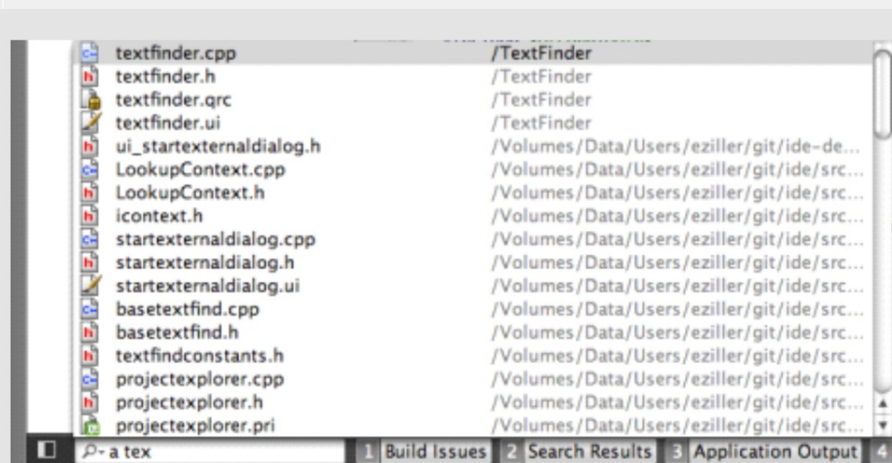
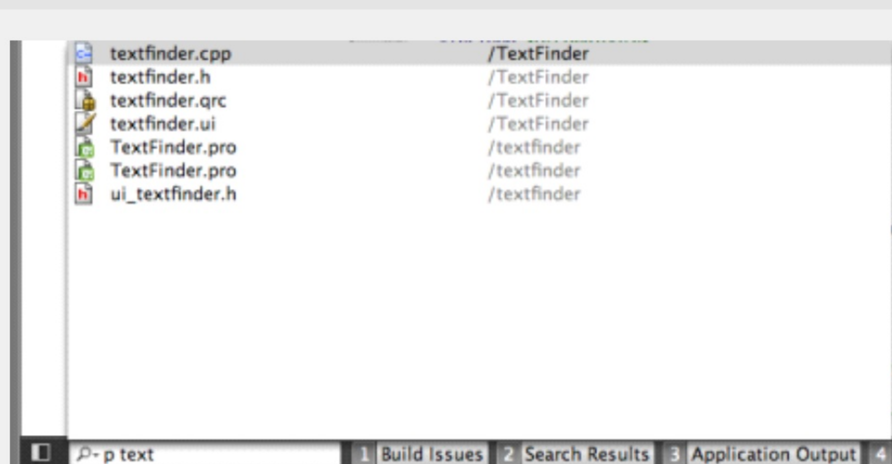


图 12-29 增加新的过滤条件

设置完成后，关闭这个对话框。定位器将按照你指定的过滤条件查找适合的信息并缓存 起来。但是定位器的前端显示还没有更新，所以你需要按图 12-28 所示的上下文菜单中的【Refresh】选项来完成后台缓存和前端显示的同步。

表 12-5 列出了常见过滤条件的操作方法以及示意的屏幕截图。

过滤条件	组合键	屏幕截图
跳转至当前文档中的某一行	依次输入 Ctrl+K, l, 空格, 代码的行号	Line 142 142 1 Build Issues 2 Search Results 3 Application Output 4
定位一个符号（类、文件名、宏等）的定义	依次输入 Ctrl+K, :, 空格, 符号的名字	QDataStream QDataStream::ByteOrder QDataStream::Status QDataStream::Version QDataStream::byteOrder () const QDataStream::device () const QDataStream::operator << (quint8) QDataStream::operator << (quint16) QDataStream::operator << (quint32) QDataStream::operator << (quint64) QDataStream::operator >> (quint64 &) QDataStream::operator >> (quint8 &) QDataStream::operator >> (quint32 &) QDataStream::operator >> (quint16 &) QDataStream::setVersion (int) QDataStream::version () const QDate : qdat 1 Build Issues 2 Search Results 3 Application Output 4
定位某一条帮助主题	依次输入 Ctrl+K, ?, 空格, 主题的名字	addKeyboardFilter addKeyClick addKeyClicks addKeyPress addKeyRelease allKeys buildKey cacheKey childKeys closeKeyboard currentKey doKeyboardAction filterKeyColumn findKey grabKeyboard grabKeyboardEvent hasKey ? key 1 Build Issues 2 Search Results 3 Application Output 4
定位一个打开的文档	依次输入 Ctrl+K, o, 空格, 文档名	Makefile /Volumes/Data/Users/eziller/git/ide-beta... extensionsystem_test.pri /Volumes/Data/Users/eziller/git/ide/src/li... optionsparser.cpp /Volumes/Data/Users/eziller/git/ide/src/li... plugindetailsview.cpp /Volumes/Data/Users/eziller/git/ide/src/li... pluginmanager.cpp /Volumes/Data/Users/eziller/git/ide/src/li... textfinder.cpp /TextFinder o 1 Build Issues 2 Search Results 3 Application Output 4

定位文件系统中的某一个文件（浏览文件系统）	依次输入 Ctrl+K，f，空格，文件名	
定位在最近载入的项目中的某一个文档	依次输入 Ctrl+K，a，空格，文档名	
定位在当前项目中的某一个文档	依次输入 Ctrl+K，p，空格，文档名	

定位一个类的定义	依次输入 Ctrl+K，c，空格，类名	
定位一个方法的定义	依次输入 Ctrl+K，m，空格，方法名	

表 12-5 常见过滤条件的操作方法和示意图

12.8 如何创建一个项目

第 1 步,创建项目 要创建一个新项目，可以依次点击菜单 【File】→【New...】→【Projects】，你可以创建下列几种项目类型:

- Qt4 Console Application – 控制台应用程序
- Qt4 Gui Application – GUI 应用程序（主要是含有界面布局的类型）
- C++ Library – C++库

这里我们创建一个基于 Qt4 Gui Application 类型的例子，如图 12-30 所示，点击【OK】 按钮进入下一步。

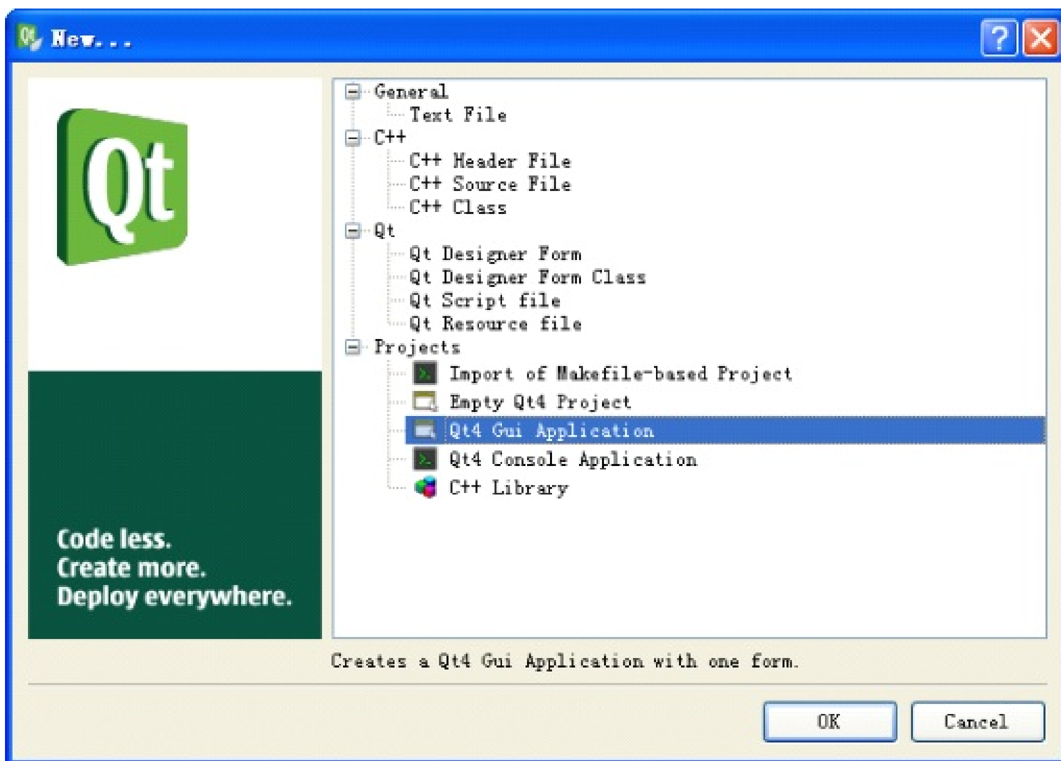


图 12-30 第 1 步 - 创建一个新的项目

第 2 步，设置项目名称和保存位置

接下来，如图 12-31 所示，我们设置项目的名称和路径。注意，项目的名称和路径中尽 量不要包含空格和其它特殊的字符，切记！设置完成后，点击 【Next】按钮进入下一步。

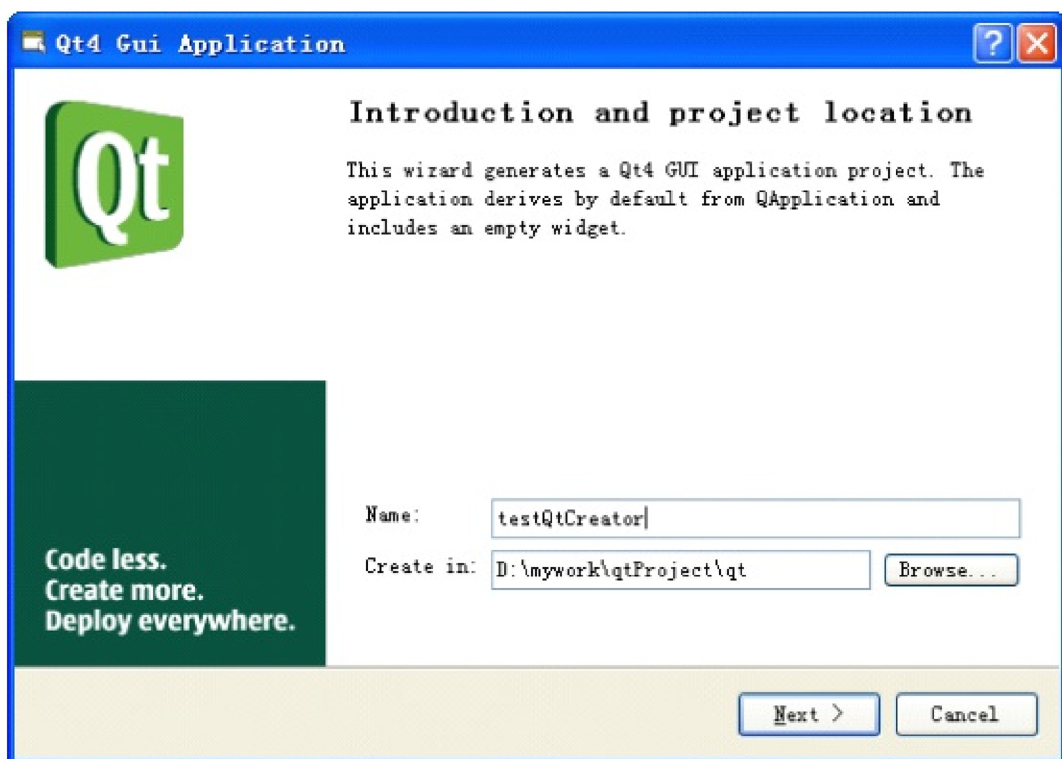


图 12-31 第 2 步 – 设置项目名称和存放位置

第 3 步，选择需要的 Qt 模块

如图 12-32 所示，在其中罗列的选项中勾选你需要的 Qt 模块，由于我们创建的是 GUI 应用程序，因此”Qt Core Moudle”和”Qt Gui Moudle”模块是默认必须选择的，其它的 可以根据需要选择。选择完后，点击 Next 按钮进入下一步。

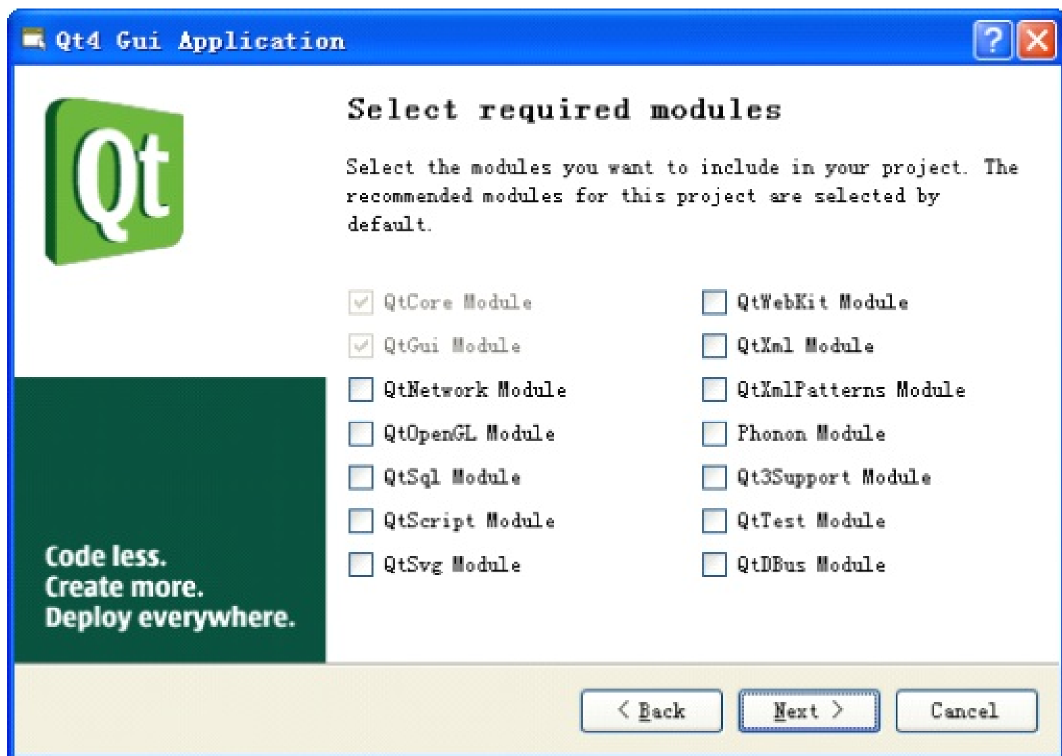


图 12-32 第 3 步 – 选择必需的 Qt 模块

第 4 步，指定类信息

如图 12-33 所示，这里最重要的是指定类的名称，类的头文件、实现文件以及 .ui 文件 的名称会随之改变。还有就是要指定你的类的基类，如 QMainWindow、QWidget 等，可以从下拉列表框中选择。设置完成后，点击 Next 按钮进入下一步。

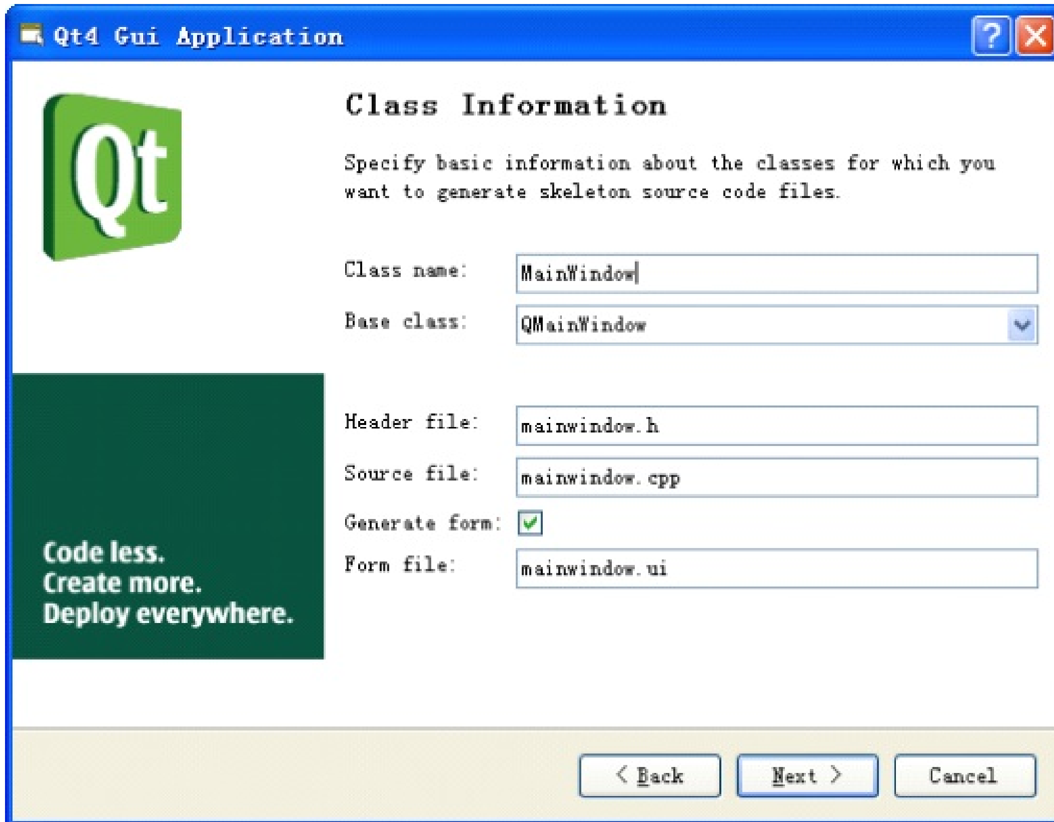


图 12-33 第 4 步 – 指定类信息

第 5 步，完成项目的创建

如图 12-34 所示，到了这一步，请仔细屏幕中列出的所有文件是否符合你的要求， 如果 不符，可以点击 Back 按钮，回去重新设置。如果确认无误，点击 Finish 按钮完成项目的创建。

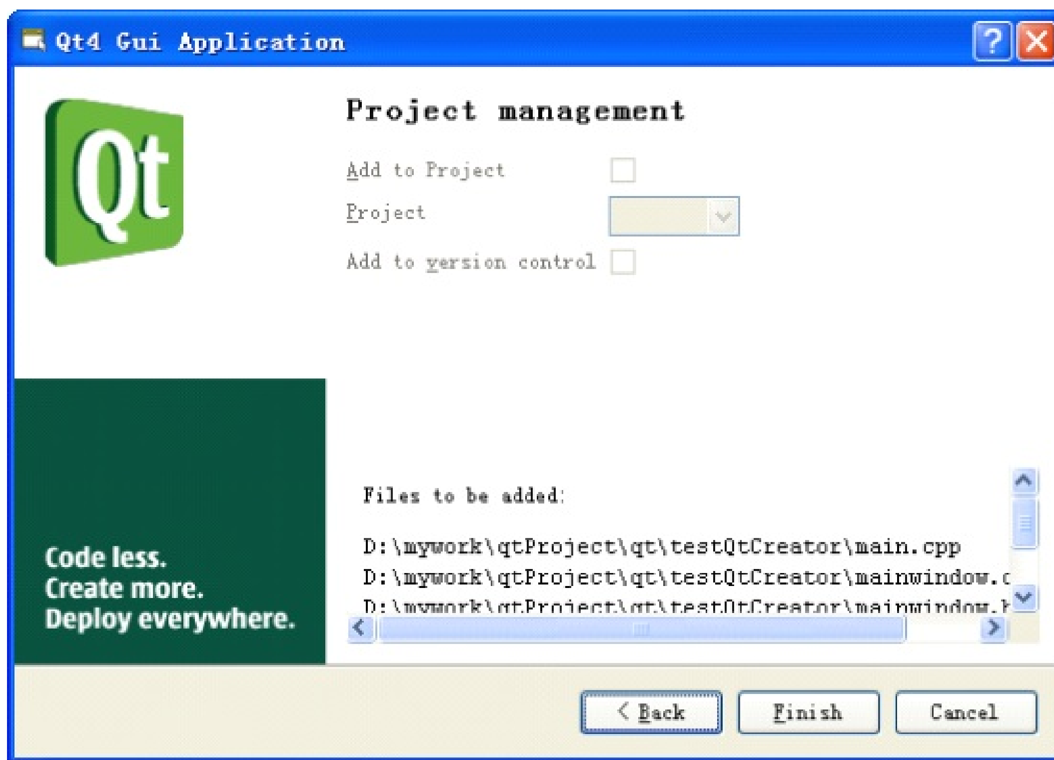


图 12-34 第 5 步 – 完成项目的创建

12.9 实例讲解

在本节中，我们以程序 textfinder 为例，向大家详细讲解使用 Qt Creator 创建应用程 序的全过程，我们将使用 Qt Creatro 创建工程和代码，并使用 Qt Designer 创建用户界面。 如果你对如何使用 Qt Designer 还不太熟悉的话，建议回头看看前几章。这个例子的运行效 果如图 12-35 所示。

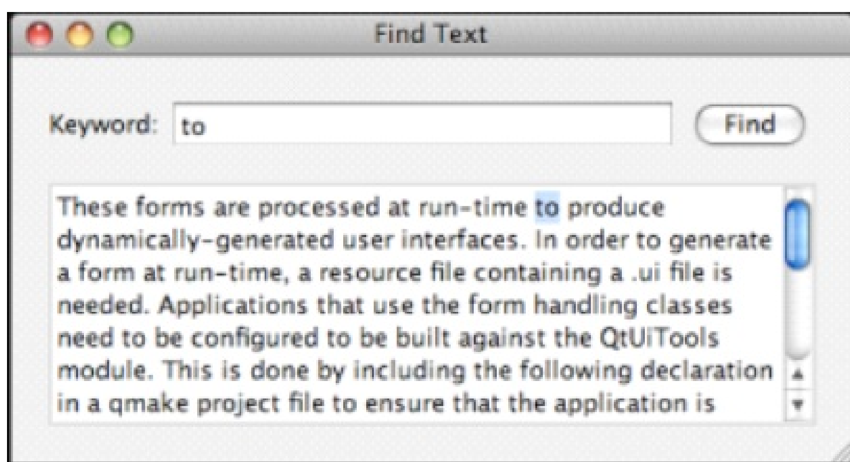


图 12-35 程序运行效果

12.9.1 程序运行内部机理

图 12-36 是笔者画的一个本程序的运行内部机理示意图， 从中可以清晰的看到各个组件 式是如何配合起来完成程序运行的。

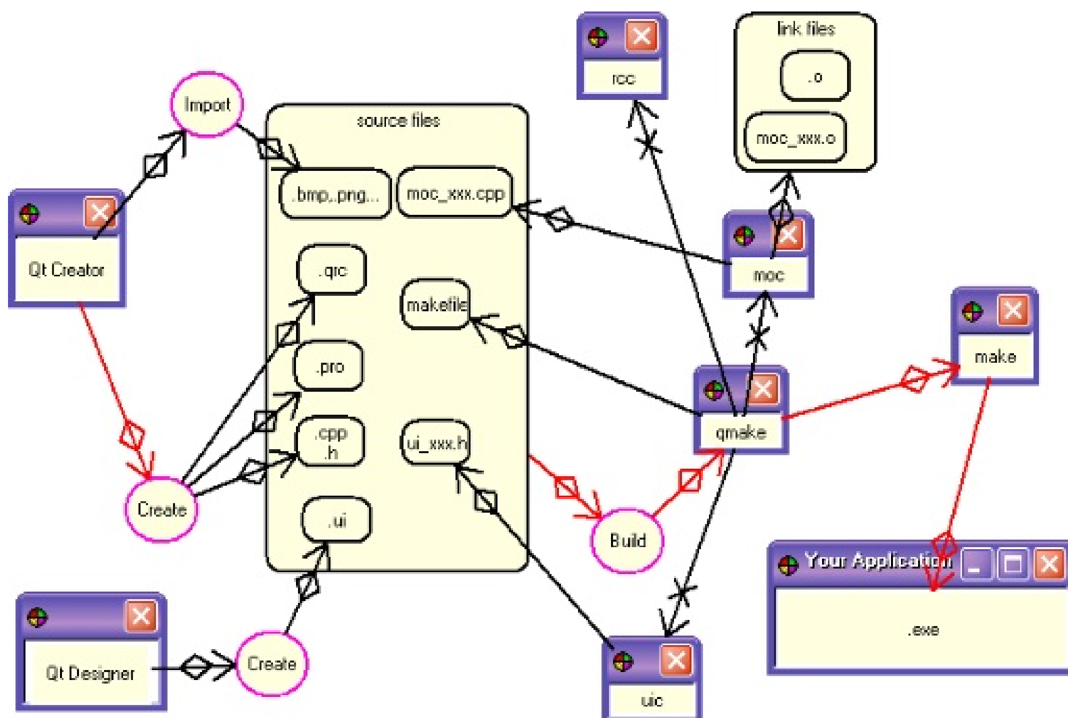


图 12-36 程序的内部机理

12.9.2 设置环境

在前面我们已经讲到，如果你是采用 SDK 安装的 Qt4，那么正常情况下，安装程序已经 自动为你设置好了所需的环境。如果你依次点击菜单【Tools】→【Options...】→【Qt4】， 却发现没有找到任何正确的 Qt 版本，那么你需要自行设置 Path 环境变量，这个步骤与平台 相关：

- 在 Windows 和 Linux 下面：依次点击菜单【Tools】→【Options...】→【Qt4】。
- 在 Mac OS X 上面：依次点击菜单【Qt4】→【Preferences】。 小贴士：如果你使用 Visual Studio 编译 Qt,再单独安装 Qt Creator，那么 Qt Creator 中 环境变量的设置与 Visual Studio 中将保持一致。

12.9.3 创建并组织项目

接下来按照上一节所述的步骤创建项目并组织好项目文件。注意这里我们要选择 QWidget 作为基类。在我们的项目中，应该包含如下文件：

- textfinder.h
- textfinder.cpp
- main.cpp
- textfinder.ui
- textfinder.pro

其中，.h 和.cpp 文件包含了程序运行所必需的基本代码，而 .pro 文件已经完成了。在接下来的步骤中，我们将使用 Qt Designer 设计界面，并添加完成功能所必须的代码。

12.9.4 设计用户界面

在你的项目浏览器（Project Explorer）中双击 textfinder.ui，将打开集成的 Qt Designer，在里面完成对用户界面的设计，并依照表 12-6 列出的内容设置各个元素的属性，完成后的情形如图 12-37 所示。

表 12-6 界面元素属性

窗口部件	名称（objectName）
QLabel	无
QLineEdit	lineEdit
QPushButton	findButton
QTextEdit	textEdit
QGridLayout	无
QVBoxLayout	无

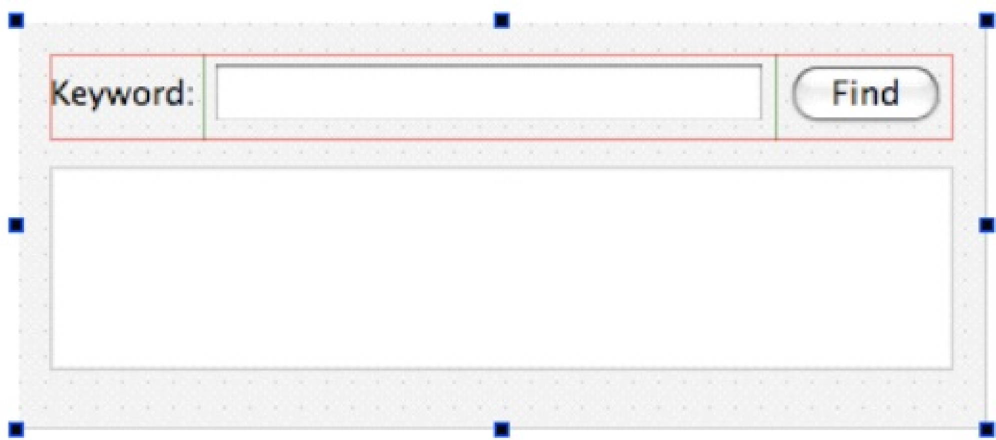


图 12-37 界面布局

该界面元素的布局方式如下：Keyword 标签和旁边的 lineEdit 以及最右边的 Find 按钮 使用 QGridLayout 组合，再与下面的 textEdit 使用 QVBoxLayout 组合。

12.9.5头文件

接下来我们看看 textfinder.h 这个头文件时怎样写的。由于我们的用户界面只有一 个，所以决定采用单继承的方式使用 .ui 文件，这就需要添加一个私有的成员变量： Ui::TextFinder *ui；我们需要添加一个私有的槽函数，以执行查找操作，它是 on_findButton_clicked()；我们还需要一个私有成员函数 loadTextFile()，用来读取并显 示我们在文本框中输入的文本文件的内容。以下是头文件中这部分的代码：

```
private slots:
    void on_findButton_clicked();
private:
    Ui::TextFinder *ui;
    void loadTextFile();
```

12.9.6实现文件

现在我们看看如何书写实现文件，这其中的关键是 loadTextFile()方法：

```
void TextFinder::loadTextFile()
{
    QFile inputFile(":/input.txt");
    inputFile.open(QIODevice::ReadOnly);
    QTextStream in(&inputFile);
    QString line = in.readAll();
    inputFile.close();
    ui-&gt;textEdit-&gt;setPlainText(line);
    QTextCursor cursor = ui-&gt;textEdit-&gt;textCursor();
    cursor.movePosition(QTextCursor::Start, QTextCursor::MoveAnchor, 1);
}
```

在上面这段代码中，我们首先使用一个 QFile 类的对象来加载文本文件，并使用 QTextStream 来读取它的内容，最后使用 setPlainText()方法来显示它。在实现文件开头要 加入下面的头文件声明：

```
#include &lt;QtCore/QFile&gt;
#include &lt;QtCore/QTextStream&gt;
```

在 on_findButton_clicked()中，我们获得了搜索的字符串并使用 find()方法在整个文本文 件中搜索该字符串，下面是实现代码：

```
void TextFinder::on_findButton_clicked()
{
    QString searchString = ui-&gt;lineEdit-&gt;text();
    ui-&gt;textEdit-&gt;find(searchString, QTextDocument::FindWholeWords);
}
```

这之后，我们需要在类的构造函数中调用 loadTextFile()这个方法，注意它应放在 setupUi()方法的后面，切记！

```
TextFinder::TextFinder(QWidget *parent)
: QWidget(parent), ui(new Ui::TextFinder)
{
    ui-&gt;setupUi(this);
    loadTextFile();
}
```

由于我们对槽函数的命名方式符合“自动关联”的规则，所以 on_findButton_clicked()槽 会被自动调用。我们在本书的前几章讲过，这是由于 uic 工具生成的 ui_textfinder.h 文件中加 入了下面这行代码的缘故，这里再次提出以加深印象。

```
QMetaObject::connectSlotsByName(TextFinder);
```

12.9.7 资源集文件

我们需要一个资源集文件（.qrc）来描述程序用到的资源，以前我们介绍的主要是如何 加入图标、图像文件，这次看看如何加入文本文件。其实方法是类似的，在项目浏览器（Project Explorer）中右键点击项目，在上下文菜单中选择【Add New ...】 →【Qt】→【Qt Resource

File】，将弹出【New Resource file】对话框。

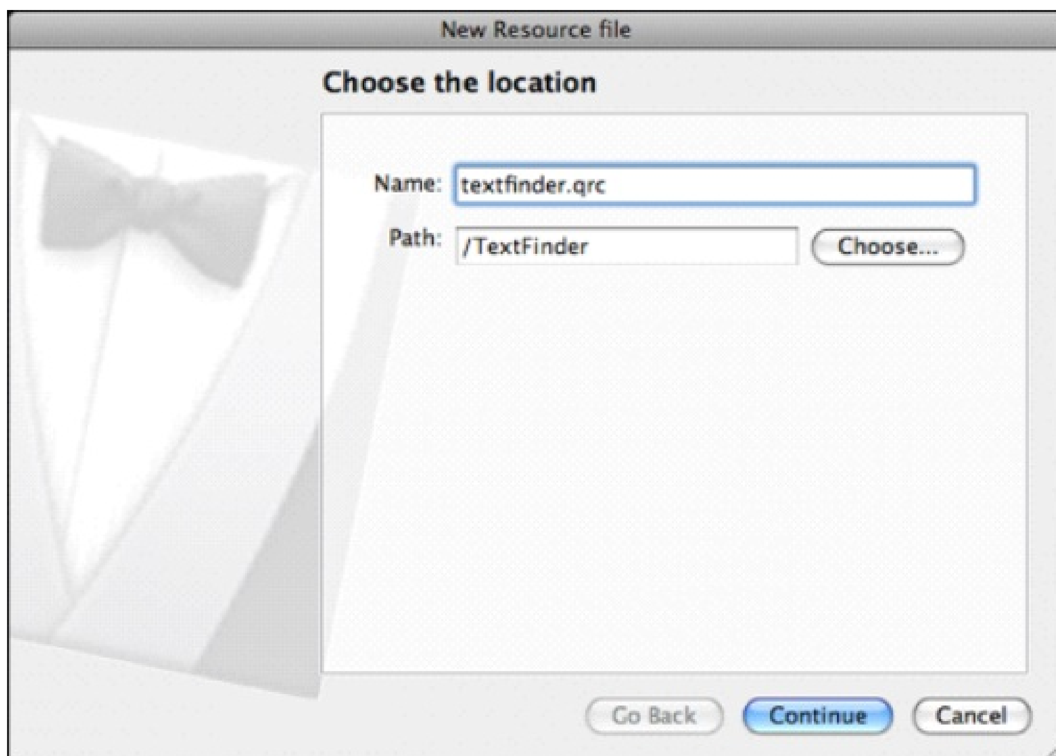


图 12-38 New Resource file 对话框

如图 12-38 所示，填入文件名字和路径，然后点击 Continue 按钮,进入下一步。

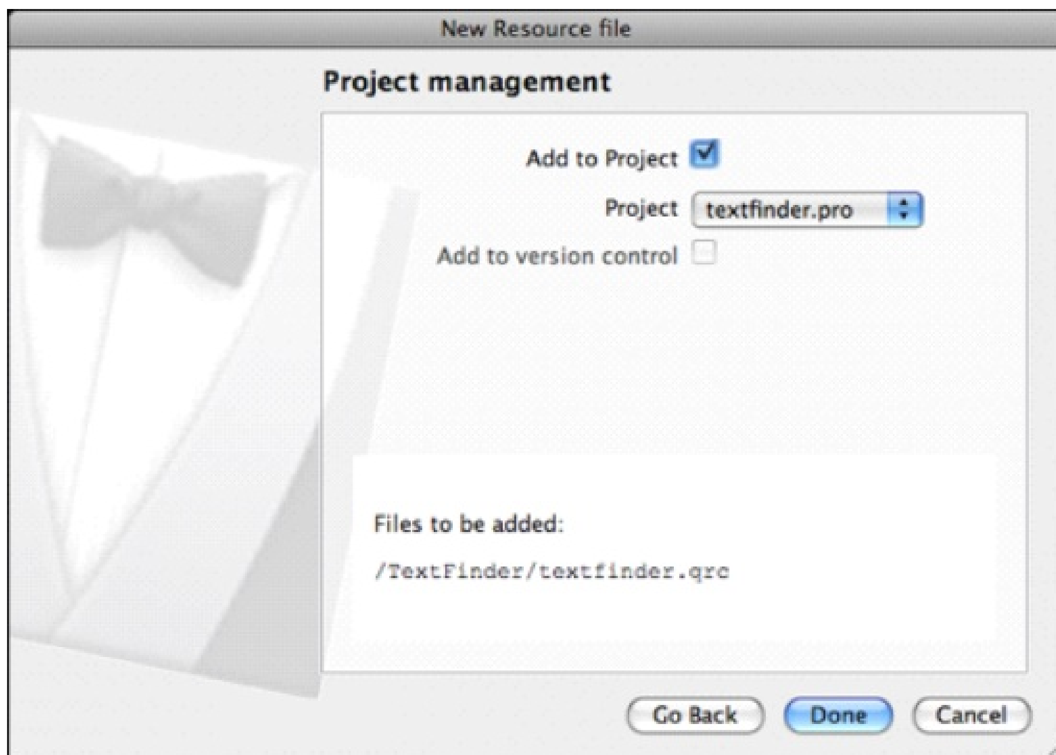


图 12-39 选择工程并加入文件

如图 12-39 所示，选中一个项目以加入资源集文件，这里是 textFinder，并确保选中【Add to Project】，然后点击【Done】按钮。

如图 12-40 所示，你的资源集文件将被资源编辑器（Resource Editor）打开并显示出 来，首先点击【Add】按钮，在下拉项中选择【Add Prefix】,这将添加一个斜线；接下来再 次点击【Add】按钮，这次选择【Add File】,找到 input.txt 文件的位置并添加它。

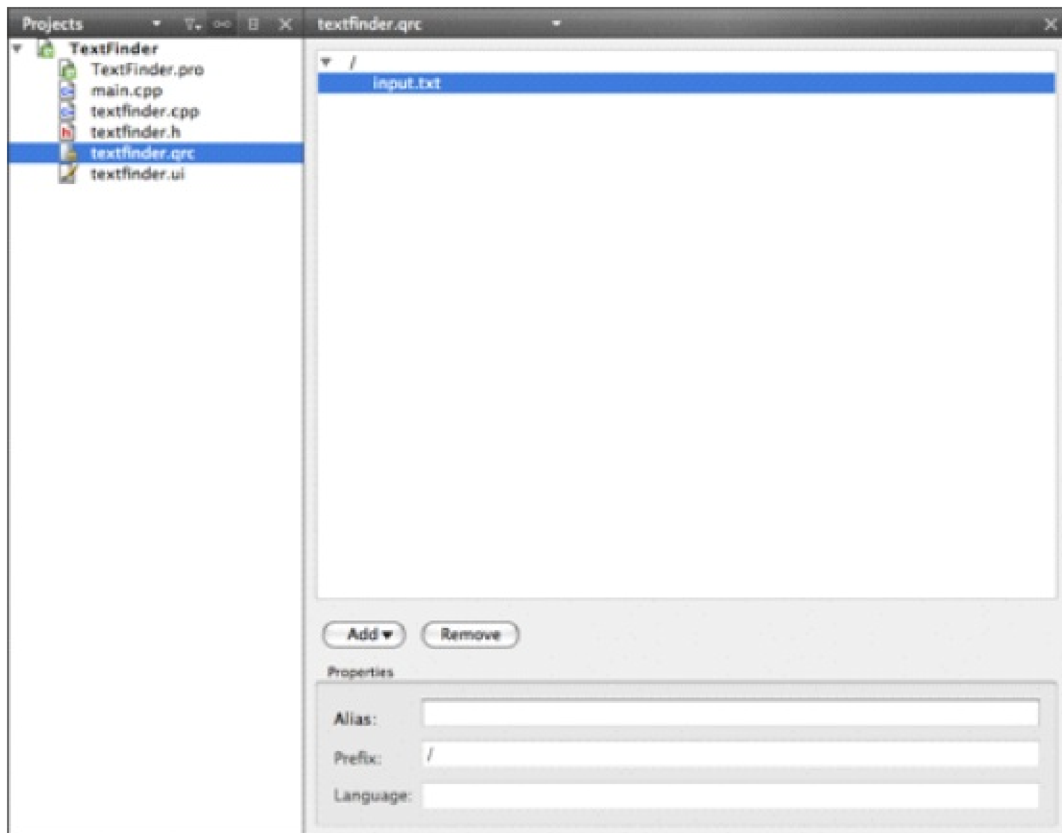


图 12-40 编辑资源集文件

12.9.8 编译运行程序

现在，所有必需的文件和准备工作都已完成，你可以按下 Ctrl+R 组合键或者点击 图标来编译运行你的程序了，程序运行的效果大致如图 12-35 所示。

12.10 使用 Qt Creator 调试程序

Qt Creator 集成了强大的调试器，提供了丰富多样的调试功能和选项，足以满足开发者 的需要。

12.10.1 调试器引擎

Qt Creator 本身并没有调试器，它必须借助其它的调试器引擎，并为它们提供了一个图 形化的前端界面。表 12-7 示出了在所支持的平台上，Qt Creator 使用的调试器。

表 12-7 Qt Creator 使用的调试器引擎

平台	编译器	调试器引擎
Linux, Unixes, Mac OS	gcc	GNU Symbolic Debugger (gdb)
Windows/MinGW	gcc	GNU Symbolic Debugger (gdb)
Windows	Microsoft Visual C++	Compiler Debugging Tools for Windows/Microsoft Console Debugger (CDB)

在 Qt Creator 中，你可以使用调试器前端界面逐行单步或逐过程调试程序，设置断点，检查堆栈中的内容，查看局部或全局变量的值等等，这些和我们常见的调试器提供的功能并无二致。而上述的原生信息，Qt Creator 会以清晰、简明的方式展现给程序员，这将使得原 本令人生畏的调试工作变得简单而有趣。

除了像堆栈查看器、局部变量和观察器、寄存器查看器等这些主流 IDE 都会提供的功能 外，Qt Creator 还提供了许多的功能以帮助开发者提高效率。由于调试器前端对 Qt 的内部 机制了如指掌，所以当程序出现问题时，它能够明晰描述症状。

表 12-8 示出了这些调试器引擎在单独安装时的一些需要注意的事项。

表 12-8 调试器引擎的相关信息

调试器引擎	注意事项
GDB（X11 平台）	需要 GDB6.8 或以上版本
GDB 或 CDB	可以从 Microsoft Developer Network 上自由下载 CDB，版本 6.10 以上，注意区分 32 位和 64 位版；（Windows 平台） 如果在 Windows 上使用 SDK 方式安装 Qt4 开源版，那么仍将使用 GDB 作为调试器引擎；如果使用 Microsoft Visual C++的编译器编译安装 Qt Creator，那么将使用 CDB 作为调试器引擎,并且默认情 况下，Qt Creator 将会检查%ProgramFiles%\Debugging Tools for Windows 这个路径下是否包含了所有需要的 调试器引擎的头文件。

12.10.2 与调试器交互

在 Debug 模式下时，Qt Creator 提供了许多的锚接窗口来辅助开发者与程序进行交互 。 常见的一些被设置为缺省可见的，不常使用的则缺省被隐藏。你可以依次点击【Debug】→【View】来配置它们的显隐。

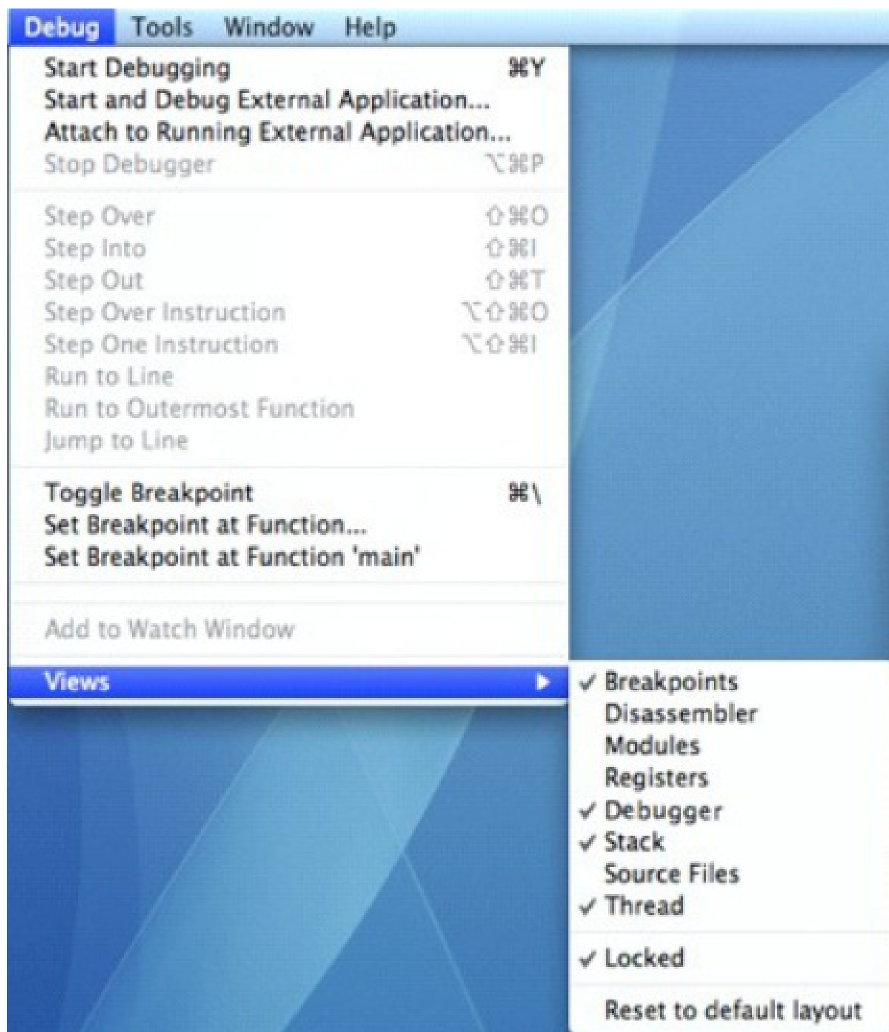


图 12-41 设置常见辅助视图的显隐

如图 12-41 所示，你可以通过点击【Locked】菜单项来锁住或解锁你的锚接窗口的布局， 就像设置这些锚接窗口的显隐一样。你的锚接窗口的位置将被 Qt Creator“记住”，下次启 动时它将根据上次的记忆来布局。

12.10.3 断点

你可以在断点视图（ Breakpoints view ）中查看断点。无论你的程序是否在运行和 调试中，断点视图都是默认并且随时可见的 。如图 12-42 所示，你可以在【Breakpoints】视图中查看断点设置的情况。图 12-43 显示了详细的断点信息。

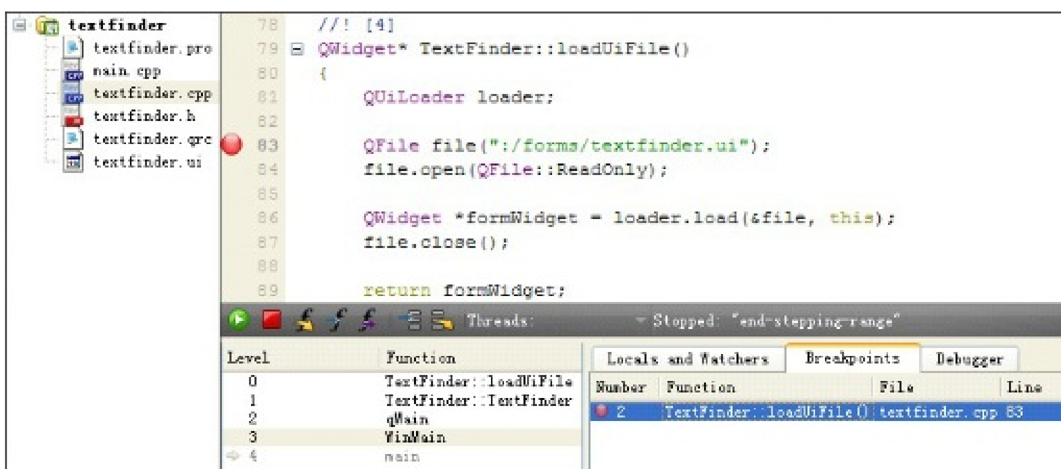


图 12-42 在【Breakpoints】视图中查看断点



图 12-43 断点的详细信息

所谓断点，就是由程序开发者设定的一系列条件，但程序以调试方式运行时，一旦符合 引发断点的“条件”，程序便中断执行，此后程序开发者便可以检视程序在运行时的状态， 继 而控制程序的运行，直至找出问题所在。

在 Qt Creator 中，我们通常可以把断点与源代码文件或者其中的某一行关联起来，也 可以把它放在某个方法的起始处（通常指定义处 ）。下面是设置断点的具体“规则”：

- 在某一行代码设置断点--在代码行行号的左边缘处点击鼠标左键或者按下 F9 键（在 Mac OS X 系统中是 F8 键）
- 在某一个函数处设置断点—依次点击菜单【Debug】 ->【Set Breakpoint at Function... 】，在其中输入函数的名字

你可以这样去掉一个断点：

- 在代码编辑器内断点标识处用鼠标左键再次点击
- 在断点视图中选中某个断点，并按下 Delete 键
- 在断点视图内点击鼠标右键，在弹出的上下文窗口中选择 【Delete Breakpoint】 小贴士：断点可以在任意时刻设置-在程序开始调试之前和正在调试之时均可。断点的设置 也会作为一部分被当前的会话所保存。

12.10.4 程序的调试运行

要在调试模式下启动运行一个 Qt 应用程序，你可以依次点击菜单项【Debug】 →【Start Debugging】，或者按下 F5 键即可。Qt Creator 将检查程序代码或设置是否有更新，并在必要时重新编译项目，然后调试器将接管并启动程序的运行。

提示：Qt 应用程序在调试模式下启动运行时，往往需要一段时间，从几秒到若干分钟 不等，这取决于你的机器的配置以及程序的复杂程度（比如应用了 QtWebKit 模块的程序可能 要多花费一些时间）。

当程序调试运行未遇到断点时，它与直接运行状态并无区别。开发者可以依次点击菜单 项【Debug】 →【Interrupt】或者直接按下如图所示的调试器状态栏上的 【Interrupt】按 钮来中断程序的运行，这与程序在运行时遇到断点而停下来的效果是一样的 ,如图 12-44 所 示。



图 12-44 调试器状态栏上的【Interrupt】按钮

当程序中断时，Qt Creator 将做如下的事情：

- 获得程序中断处在堆栈中的地址
- 获得局部变量的值
- 检视并更新观察器（Watchers）视图内容

更新 Registers 、Modules 以及 Disassembler 视图 这时我们可以在 Debugger 视图中检视到程序更为详细的状态。

要结束调试状态，可以按下 Shift+F5 键。按下 F10 键可以进入逐行调试状态，按下 F11 键进入逐过程调试状态，按下 F5 键可以使程序运行到下一个断点处，如果后面没有断点了， 程序将完整的运行起来，这种情况仍然是在调试状态下的运行。

12.10.5 堆栈视图（Stack View）

当被调试的程序在断点处中断时，Qt Creator 将在堆栈视图中显示出程序到达断点处之 前所经历的那些函数。这些函数对应到被称作是“堆栈框架节点”，每一个节点对应一个函数。 如图 12-45 所示，Qt Creator 显示了这些函数所在的文件名、在源代码里面的行号。

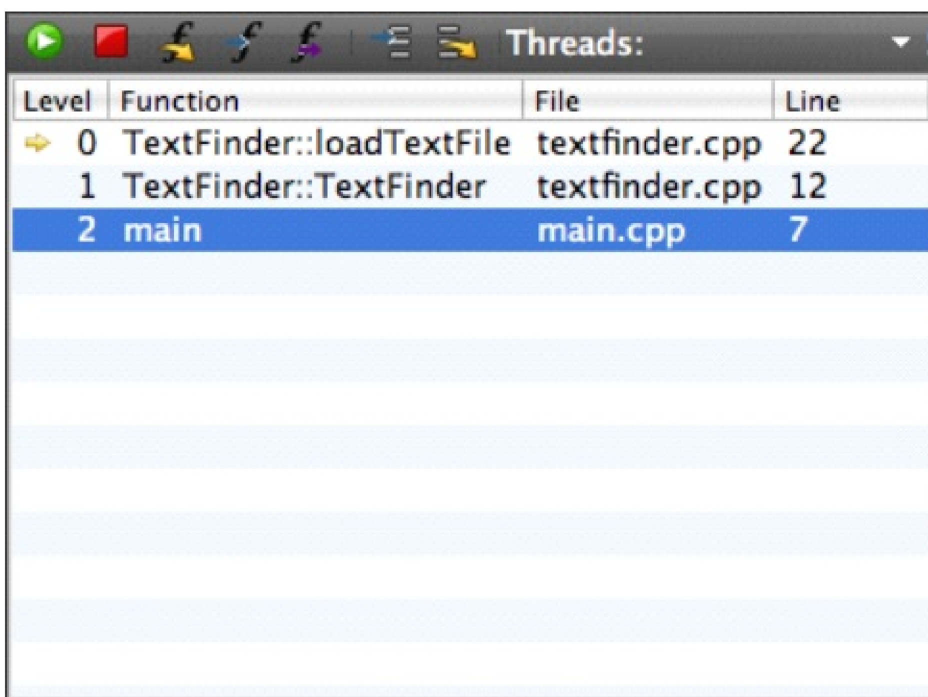


图 12-45 堆栈视图

有些情况下，不是所有的框架节点都能够准确的对应到源代码中的一个位置， 因而也就 没有相应的调试信息，这些调试框架将被灰色显示。

当你在堆栈视图里面显示的某一行处使用鼠标左键双击时， 代码编辑器将跳转至相应的 代码行，Qt Creator 将更新局部变量和观察器视图，就像把断点设置到这个地方而程序正好 在这里中断时的情形一样。

12.10.6 线程视图（Thread View）

当我们调试一个多线程应用程序时，如图 12-46 所示，线程视图（thread view）和调 试器状态栏上的"Thread"组合框（见图 12-47）被用来在不同线程间切换，这时堆栈视图（stack view）也将会随着做出相应的调整。

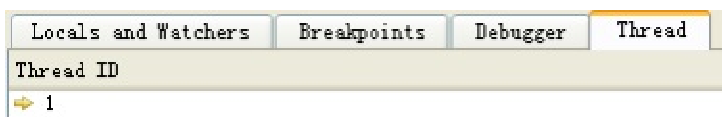


图 12-46 线程视图



图 12-47 调试器状态栏上的 Thread 组合框

12.10.7 局部变量和观察器视图（Locals and Watchers View）

当程序在调试器的控制下中断时，Qt Creator 将取得堆栈里面的最上层框架节点的相关 信息并把它们显示在局部变量和观察器视图里面。

局部变量和观察器视图通常由一个树形结构组成，里面有许多的一级节点，第二级节点 等等层次的数据，比如数据结构和类等信息就不是显示在第一级节点里面的。要查看更为详 细的信息，可以逐级点开这些节点前面的 “+”号。

你也可以在局部变量和观察器中更改变量的内容（比如常见的 int 和 float 值），以界 定你想确定的变量值的限值。这可以通过双击 ”Value”栏，并在可编辑区填入你的新的取 值，然后按下回车键（Enter 或 Return 键），之后再接着调试程序。

小贴士：你对观察器里面项目的设置将被保存到这次会话里面，下次打开对话时，这些设置仍然有效。

12.10.8 模块视图（Modules Views）

默认情况下，模块视图也是不显示的。它的主要作用是使开发者了解程序中用到了那些模块，严格意义上来说，它不应该是在调试模式下才有的功能。一个常见的模块视图如图12-48所示。

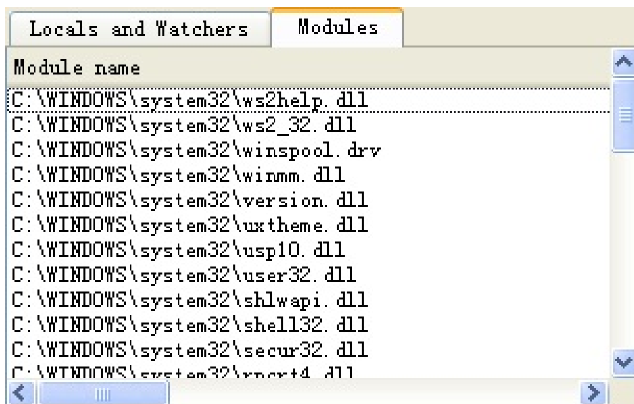


图 12-48 模块视图

12.10.9 反汇编和视图（Disassembler View）和寄存器视图（Registers View）

默认情况下，反汇编视图和寄存器视图是隐藏的。反汇编视图显示了断点处所在的函数的反汇编代码，如图 12-49 所示；寄存器视图显示了当前 CPU 的寄存器的状态，如图 12-50所示，当你需要对程序的底层（与系统硬件接触）进行检视和控制时，这两个视图尤其有用。当我们使用逐过程调试的方法时，经常会用到它们。

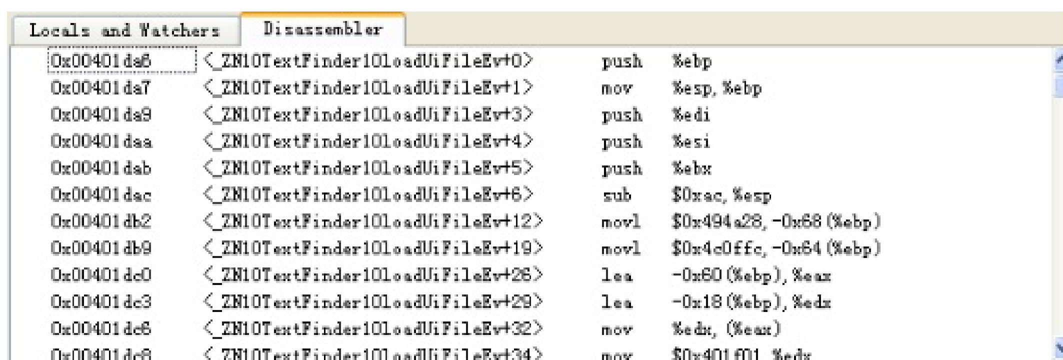


图 12-49 反汇编视图

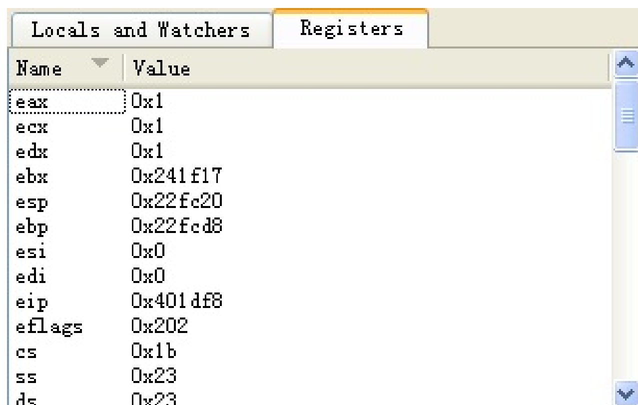


图 12-50 寄存器视图

12.10.10 程序调试实例

在我们的这个 TextFinder 例子里面，我们要使用 QString 读取一个文本文件，然后再 用一个 QTextEdit 把它的内容显示出来。在其中我们定义了一个 QString 类型的变量 line， 然后在附近设置一个断点，用来查看 line 变量的内容，请大家跟着我的步骤进行调试。

```
 94  void TextFinder::loadTextFile()
 95  {
 96      QFile inputFile(":/forms/input.txt");
 97      inputFile.open(QIODevice::ReadOnly);
 98      QTextStream in(&inputFile);
 99      QString line = in.readAll();
100      inputFile.close();
101
102      ui_textEdit->append(line);
103      ui_textEdit->setUndoRedoEnabled(false);
104      ui_textEdit->setUndoRedoEnabled(true);
105  }
```

图 12-51 设置断点

首先是设置断点，如图 12-51 所示，将光标移动到选定的位置，按下 F9 键，或者在行号前点击鼠标左键完成断点的设置。然后按下 F5 键， 启动调试。

如图 12-52 所示，在调试模式（Debug Mode）下,我们可以在断点视图（Breakpoints View）中查看已经设置的断点情况。要取消断点，可以再次点击 F9 键。

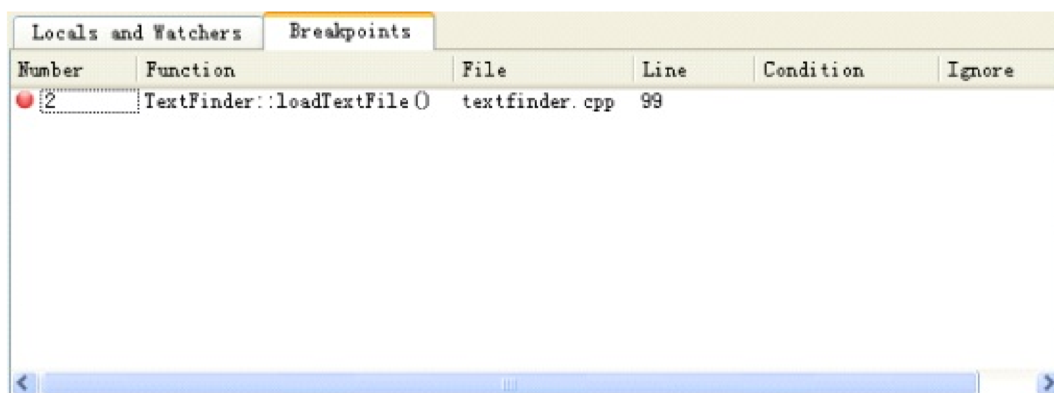


图 12-52 查看断点设置情况

可以在局部变量和观察器视图中查看变量的内容，如图 12-53 所示，显示了 line 等变 量的内容。

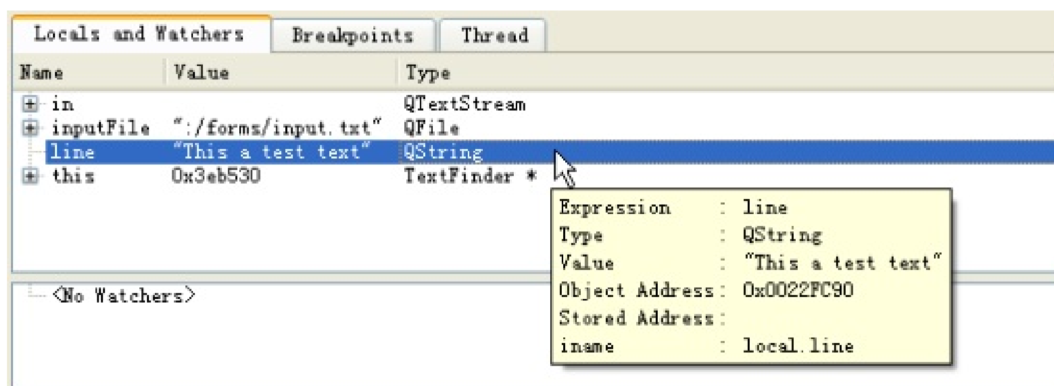


图 12-53 查看 line 变量的内容

下面是我们的程序中的槽函数 on_findButton_clicked()的代码，我们将修改代码中的 部分内容，形成一个小的逻辑错误，然后示范调试的步骤。原始正确的代码如下：

```
QString searchString = ui_lineEdit-&gt;text();
QTextDocument *document = ui_textEdit-&gt;document();
bool found = false;
if (isFirstTime == false)
    document-&gt;undo();
if (searchString == "")
{
    QMessageBox::information(this, tr("Empty Search Field"),
        "The search field is empty. Please enter a word and click Find.");
}
else
{
    QTextCursor highlightCursor(document);
    QTextCursor cursor(document);
    cursor.beginEditBlock();
    QTextCharFormat plainFormat(highlightCursor.charFormat());
    QTextCharFormat colorFormat = plainFormat;
    colorFormat.setForeground(Qt::red);
    while (!highlightCursor.isNull() && !highlightCursor.atEnd())
    {
        highlightCursor = document-&gt;find(searchString, highlightCursor,
        QTextDocument::FindWholeWords);
        if (!highlightCursor.isNull())
        {
            found = true;
            highlightCursor.movePosition(QTextCursor::WordRight,
            QTextCursor::KeepAnchor);
            highlightCursor.mergeCharFormat(colorFormat);
        }
    }
    cursor.endEditBlock();
    isFirstTime = false;
    if (found == false)
    {
        QMessageBox::information(this, tr("Word Not Found"),
            "Sorry, the word cannot be found.");
    }
}
```

我们将第 6 行改为：

```
if (searchString != "")
```

大家注意，改动之处是把 比较运算符==变成了!=，这时运行程序，无论你输入任何有效的字符，程序的运行结果总是与你的预期相反。那么就需要设置断点，调试程序了。在第 一行处按下 F9 键，然后按下 F5 键开始调试，如图 12-54 所示，使用鼠标点击调试器工具栏 上的常用按钮或者按下对应的快捷键，执行逐行调试或逐过程调试均可，如图 11-54 所示。 程序单步执行到第 6 行时，你将会发现这个逻辑错误。把它更正过来，再次调试程序即可 。

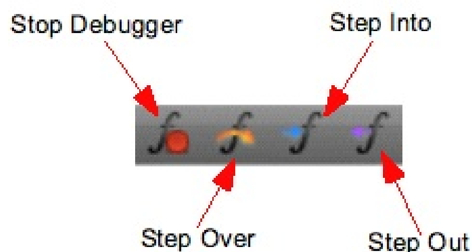


图 12-54 调试器工具栏

至此，关于 Qt Creator 的使用的介绍就结束了。要想掌握好 Qt Creator，使之成为你 的左膀右臂，就需要多实践，多总结。

12.11 问题与解答

问：如何在各个模式间快速切换？

答：可以使用 Ctrl+1, Ctrl+2 这样的组合快捷键来切换模式。 问：如何在命令行使用 Qt Creator 并打开工程？ 可以通过在命令行输入如下命令来调用 Qt Creator 并打开工程： Qt Creator xxx.pro

问：如何显隐边栏（sidebar）？

在 Edit 和 Debug 模式下，你可以通过按下 Ctrl+0 组合键来显隐边栏。 问：Qt Creator 是否支持不使用 qmake 创建的工程呢？

答：从 Qt Creator 1.1 版（包含在 Qt 4.5.1 中）开始，就可以支持其它的通用工程了（即不是使用 qmake 或 CMake 创建的工程），这时候，Qt Creator 将仅仅作为一个代码编辑 器使用。你可以在 Project Settings 页面下设置你要使用的编译系统。

问：我的程序代码没有问题（经过检查了），为什么我在调试时发现某些变量的值在有 时候变得非常奇怪，而到最后又好了。

答：gdb，以及采用它作为调试器引擎的 Qt Creator 的相应版本对 Linux 和 Mac OS X 平台上应用程序的编译做了优化。由于这些优化措施可能会导致函数过程的重组甚至会移动 某些局部变量在堆栈或堆中的位置。所以，你在局部变量和观察器视图中可能会看到某些远 离期望值的代码。，由于 gcc 对运行时正在初始化的局部变量并没有提供足够的调试信息， 所 以有时候你在 Qt Creator 的局部变量和观察器中查看某个变量的值时， 发现系统提示“超出 范围”。

12.12 总结与提高

Qt Creator 是 Qt4 应用开发中的首选 IDE。本章采用图文结合的形式，全面讲解了 Qt Creator 的使用方法和步骤。这些内容都是在项目开发中经常用到的必会技能，希望读者朋 友熟练掌握。

Qt Creator 还有许多高级的功能，比如如何使用 CMake（而不是使用 qmake）构建项目、如何在其中使用版本控制软件等等，它们已经超出了本书的范围，有兴趣的读者可以有针对性学习这些内容。

第 13 章 Qt 核心机制与原理

本章重点

- 了解 Qt 和 C++的关系
- 掌握 Qt 的信号/槽机制的原理和使用方法
- 了解 Qt 的元对象系统
- 掌握 Qt 的架构
- 理解 Qt 的事件模型，掌握其使用的时机

信号与槽、元对象系统、事件模型是 Qt 机制的核心，如果您想要掌握 Qt 编程，就需 要对它们有比较深入的了解。本章重点介绍了信号与槽的基本概念和用法、元对象系统、 Qt 的事件模型，以及它们在实际使用过程中应注意的一些问题。

13.1 Qt 对标准 C++的扩展

标准 C＋＋对象模型为面向对象编程提供了有效的实时支持，但是它的静态特性在一 些领域中表现的不够灵活。事实上， GUI 应用程序往往对实时性和灵活性都有着很高的要 求。Qt 通过其改进的对象模型在保持 C＋＋执行速度的同时提供了所需要的灵活性。

Qt 相对于标准 C＋＋增添的特性主要有以下体现：

- 支持对象间通信信号与槽机制
- 支持可查询和可设计的动态对象属性机制
- 事件和事件过滤器
- 国际化支持
- 支持多任务的定时器
- 支持按层检索的对象树
- 受保护指针
- 动态类型转换

这些内容是 Qt 核心机制的重要组成部分，在下面的章节中，笔者将有选择的向大家介 绍它们。

13.2 信号与槽

信号和槽机制是 Qt 的核心机制之一，要掌握 Qt 编程就需要对信号和槽有所了解。信号和槽是一种高级接口，它们被应用于对象之间的通信，它们是 Qt 的核心特性，也是 Qt不同于其它同类工具包的重要地方之一。

在我们所了解的其它 GUI 工具包中，窗口小部件(widget)都有一个回调函数用于响应 它们触发的动作，这个回调函数通常是一个指向某个函数的指针。在 Qt 中用信号和槽取代 了上述机制。

1.信号（signal）

当对象的状态发生改变时，信号被某一个对象发射（ emit）。只有定义过这个信号的类或者其派生类能够发射这个信号。当一个信号被发射时，与其相关联的槽将被执行，就象一个正常的函数调用一样。信号-槽机制独立于任何 GUI 事件循环。只有当所有的槽正确返 回以后，发射函数（emit）才返回。

如果存在多个槽与某个信号相关联，那么，当这个信号被发射时，这些槽将会一个接 一个地被执行，但是它们执行的顺序将会是不确定的，并且我们不能指定它们执行的顺序。

信号的声明是在头文件中进行的，并且 moc 工具会注意不要将信号定义在实现文件 中。Qt 用 signals 关键字标识信号声明区，随后即可声明自己的信号。 例如，下面定义了 几个信号：

```
signals:
void yourSignal();
void yourSignal(int x);
```

在上面的语句中，signals 是 Qt 的关键字。接下来的一行 void yourSignal(); 定义了信号 yourSignal，这个信号没有携带参数；接下来的一行 void yourSignal(int x);定义 了信号 yourSignal(int x)，但是它携带一个整形参数，这种情形 类似于重载。

注意，信号和槽函数的声明一般位于头文件中，同时在类声明的开始位置必须加上 Q_OBJECT 语句，这条语句是不可缺少的，它将告诉编译器在编译之前必须先应用 moc 工具 进行扩展。关键字 signals 指出随后开始信号的声明，这里 signals 用的是复数形式而非 单数，siganls 没有 public、private、protected 等属性，这点不同于 slots。另外， signals、slots 关键字是 QT 自己定义的，不是 C++中的关键字。

还有，信号的声明类似于函数的声明而非变量的声明，左边要有类型，右边要有括 号，如果要向槽中传递参数的话，在括号中指定每个形式参数的类型，当然，形式参数的个 数可以多于一个。

从形式上讲,信号的声明与普通的 C++函数是一样的，但是信号没有定义函数实现。另 外，信号的返回 类型都是 void，而 C++函数的返回值可以有丰富的类型。

注意，signal 代码会由 moc 自动生成，moc 将其转化为标准的 C++语句，C++预处理 器会认为自己处理的是标准 C++源文件。所以大家不要在自己的 C++实现文件实现 signal。

2.槽（slot）

槽是普通的 C++成员函数，可以被正常调用，不同之处是它们可以与信号（ signal）相 关联。当与其关联的信号被发射时，这个槽就会被调用。槽可以有参数，但槽的参数不能有 缺省值。

槽也和普通成员函数一样有访问权限。槽的访问权限决定了谁可以和它相连。 通常， 槽也分为三种类型，即 public slots、private slots 和 protected slots。

public slots：在这个代码区段内声明的槽意味着任何对象都可将信号与之相连接。 这对于组件编程来说非常有用：你生成了许多对象，它们互相并不知道，把它们的信号和槽 连接起来，这样信息就可以正确地传递，并且就像一个小孩子喜欢玩耍的铁路轨道上的火车 模型，把它打开然后让它跑起来。

protected slots：在这个代码区段内声明的槽意味着当前类及其子类可以将信号与之 相关联。这些槽只是类的实现的一部分，而不是它和外界的接口。

private slots：在这个代码区段内声明的槽意味着只有类自己可以将信号与之相关 联。这就是说这些槽和这个类是非常紧密的，甚至它的子类都没有获得连接权利这样的信 任。

通常，我们使用 public 和 private 声明槽是比较常见的，建议尽量不要使用 protected 关键字来修饰槽的属性。此外，槽也能够声明为虚函数。

槽的声明也是在头文件中进行的。例如，下面声明了几个槽：

```
public slots:
void yourSlot();
void yourSlot(int x);
```

注意，关键字 slots 指出随后开始槽的声明，这里 slots 用的也是复数形式。

3.信号与槽的关联

槽和普通的 C++成员函数几乎是一样的－可以是虚函数；可以被重载；可以是共有的、 保护的或是私有的，并且也可以被其它 C++成员函数直接调用；还有，它们的参数可以是任 意类型。唯一不同的是：槽还可以和信号连接在一起，在这种情况下，每当发射这个信号的 时候，就会自动调用这个槽。

connect()语句看起来会是如下的样子：

```
connect(sender,SIGNAL(signal),receiver,SLOT(slot));
```

这里的 sender 和 receiver 是指向 QObject 的指针，signal 和 slot 是不带参数的函数 名。实际上，SIGNAL()宏和 SLOT()会把它们的参数转换成相应的字符串。

到目前为止，在已经看到的实例中，我们已经把不同的信号和不同的槽连接在了一 起。但这里还需要考虑一些其他的可能性。

(1) 一个信号可以连接多个槽

```
connect(slider,SIGNAL(valueChanged(int)),spinBox,SLOT(setValue(int)));
connect(slider,SIGNAL(valueChanged(int)),this,SLOT(updateStatusBarIndicator(int)));
```

在发射这个信号的时候，会以不确定的顺序一个接一个的调用这些槽。

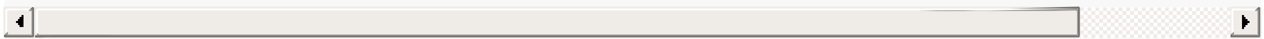
(2) 多个信号可以连接同一个槽

```
connect()
```

无论发射的是哪一个信号，都会调用这个槽。

(3) 一个信号可以与另外一个信号相连接

```
connect(lineEdit,SIGNAL(textChanged(const Qstring &)),this,SIGNAL(updateRecord(const Qstr
```



当发射第一个信号时，也会发射第二个信号。除此之外，信号与信号之间的连接和信 号与槽之间的连接是难以区分的。

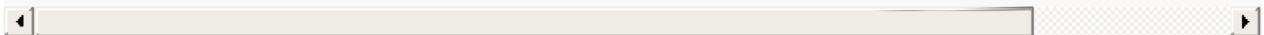
(4) 连接可以被移除

```
disconnect(lcd,SIGNAL(overflow()),this,SLOT(handleMathError()));
```

这种情况较少用到，因为当删除对象时， Qt 会自动移除和这个对象相关的所有连接。

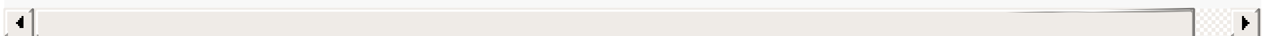
(5) 要把信号成功连接到槽（或者连接到另外一个信号），它们的参数必须具有相同的顺序 和相同的类型

```
connect(ftp,SIGNAL(rawCommandReply(int,const QString&)),this,SLOT(processReply(int,const
```



(6) 如果信号的参数比它所连接的槽的参数多，那么多余的参数将会被简单的忽略掉

```
connect(ftp,SIGNAL(rawCommandReply(int,const Qstring &)),this,SLOT(checkErrorCode(int)));
```



还有，如果参数类型不匹配，或者如果信号或槽不存在，则当应用程序使用调试模式 构建后，Qt 会在运行时发出警告。与之相类似的是，如果在信号和槽的名字中包含了参数 名，Qt 也会发出警告。

信号和槽机制本身是在 QObject 中实现的，并不只局限于图形用户界面编程中。这种 机制可以用于任何 QObject 的子类中。

当指定信号 signal 时必须使用 Qt 的宏 SIGNAL()，当指定槽函数时必须使用宏 SLOT()。如果发射者与接收者属于同一个对象的话，那么在 connect 调用中接收者参数可 以省略。

例如，下面定义了两个对象：标签对象 label 和滚动条对象 scroll，并将 valueChanged()信号与标签对象的 setNum()相关联，另外信号还携带了一个整形参数，这样标签总是显示滚动条所处位置的值。

```
QLabel *label = new QLabel;
QScrollBar *scroll = new QScrollBar;
QObject::connect( scroll, SIGNAL(valueChanged(int)),
label, SLOT(setNum(int)) );
```

4.信号和槽连接示例

以下是 QObject 子类的示例：

```
class BankAccount : public QObject
{
    Q_OBJECT
public:
    BankAccount() { curBalance = 0; }
    int balance() const { return curBalance; }
public slots:
    void setBalance(int newBalance);
signals:
    void balanceChanged(int newBalance);
private:
    int currentBalance;
};
```

与多数 C++ 类的风格类似，BankAccount 类拥有构造函数、balance() “读取”函数 和 setBalance() “设置”函数。它还拥有 balanceChanged() 信号，帐户余额更改时将 发出此信号。发出信号时，与它相连 的槽将被执行。

Set 函数是在公共槽区中声明的，因此它是一个槽。槽既可以作为成员函数，与其他 任何函数一样调用，也可以与信号相连。以下是 setBalance() 槽的实现过程：

```
void BankAccount::setBalance(int newBalance)
{
    if (newBalance != currentBalance)
    {
        currentBalance = newBalance;
        emit balanceChanged(currentBalance);
    }
}
```

语句 emit balanceChanged(currentBalance);将发出 balanceChanged() 信号，并使 用当前新余额作为其参数。

关键字 emit 类似于“signals”和“slots”，由 Qt 提供，并由 C++ 预处理器转换成标准 C++ 语句。

以下示例说明如何连接两个 BankAccount 对象：

```
BankAccount x, y;
connect(&x, SIGNAL(balanceChanged(int)), &y, SLOT(setBalance(int)));
x.setBalance(2450);
```

当 x 中的余额设置为 2450 时，系统将发出 balanceChanged() 信号。y 中的 setBalance() 槽收到此信号后，将 y 中的余额设置为 2450。一个对象的信号可以与多个 不同槽相连，多个信号也可以与特定对象中的某一个槽相连。参数类型相同的信号和槽可以 互相连接。槽的参数个数可以少于信号的参数个数，这时多余的参数将被忽略。

5.需要注意的问题

信号与槽机制是比较灵活的，但有些局限性我们必须了解，这样在实际的使用过程中才能够做到有的放矢，避免产生一些错误。下面就介绍一下这方面的情况。

(1) 信号与槽的效率是非常高的，但是同真正的回调函数比较起来，由于增加了灵活 性，因此在速度上还是有所损失，当然这种损失相对来说是比较小的，通过在一台 i586- 133 的机器上测试是 10 微秒（运行 Linux），可见这种机制所提供的简洁性、灵活性还是 值得的。但如果我们要追求高效率的话，比如在实时系统中就要尽可能的少用这种机制。

(2) 信号与槽机制与普通函数的调用一样，如果使用不当的话，在程序执行时也有可能 产生死循环。因此，在定义槽函数时一定要注意避免间接形成无限循环，即在槽中再次发射 所接收到的同样信号。

(3) 如果一个信号与多个槽相关联的话，那么，当这个信号被发射时，与之相关的槽被 激活的顺序将是随机的，并且我们不能指定该顺序。

(4) 宏定义不能用在 signal 和 slot 的参数中。

(5) 构造函数不能用在 signals 或者 slots 声明区域内。

(6) 函数指针不能作为信号或槽的参数。

(7) 信号与槽不能有缺省参数。

(8) 信号与槽也不能携带模板类参数。

6.小结

从 QObject 或其子类(例如 Qwidget)派生的类都能够使用信号和槽机制。这种机制本身 是在 QObject 中实现的，并不只局限于图形用户界面编程中：当对象的状态得到改变时， 它可以某种方式将信号发射(emit)出去，但它并不了解是谁在接收这个信号。槽被用于接收 信号，事实上槽是普通的对象成员函数。槽也并不了解是否有任何信号与自己相连接。而 且，对象并不了解具体的通信机制。这实际上是 “封装”概念的生动体现，信号与槽机制 确保了 Qt 中的对象被当作软件的组件来使用,体现了“软件构件化”的思想。

13.3 元对象系统

Qt 的元对象系统是一个基于标准 C＋＋的扩展，能够使 C＋＋更好的适应真正的组件 GUI 编程。它为 Qt 提供了支持对象间通信的信号与槽机制、实时类型信息和动态属性系统等方面的功能。

元对象系统在 Qt 中主要有以下三部分构成：QObject 类、Q_OBJECT 宏和元对象编译 器 moc。

1.元对象系统机制

Qt 的主要成就之一是使用了一种机制对 C++进行了扩展，并且使用这种机制创建了独 立的软件组件。这些组件可以绑定在一起，但任何一个组件对于它所要连接的组件的情况事 先都不了解。

这种机制称为元对象系统（meta-object system），它提供了关键的两项技术：信号－槽以及内省（introspection）。内省功能对于实现信号和槽是必需的，并且允许应用程 序的开发人员在运行时获得有关 QObject 子类的“元信息”（meta-information），包括 一个含有对象的类名以及它所支持的信号和槽的列表。这一机制也支持属性（广泛用于 Qt 设计师中）和文本翻译（用于国际化），并且它也为 QtScirpt 模块奠定了基础。

标准 C++没有对 Qt 的元对象系统所需要的动态元信息提供支持。 Qt 通过提供一个独立 的 moc 工具解决了这个问题，moc 解析 Q_OBJECT 类的定义并且通过 C++函数提供可供使用的信息。由于 moc 使用纯 C++来实现它的所有功能，所以 Qt 的元对象系统可以在任意 C++编译器上工作。

这一机制是这样工作的：

(1) Q_OBJECT 宏声明了在每一个 QObject 子类中必须实现的一些内省函数，如 metaObject()、QMetaObject::className()、tr()、qt_metacall()，以及其它一些函数。

(2) Qt 的 moc 工具生成了用于由 Q_OBJECT 声明的所有函数和所有信号的实现。

(3) 像 connect()和 disconnect()这样的 QObject 的成员函数使用这些内省函数来完成 它们的工作。

由于所有这些工作都是由 qmake 和 QObject 类自动处理的，所以很少需要再去考虑这 些事情，如果想进一步了解的话，也可以阅读一下有关 QMetaObject 类的文档和由 moc 生 成的 C++源代码文件，可以从中看出这些实现工作是如何进行的。

2.元对象工具（moc）

Qt 的信号和槽机制是采用标准 C++ 来实现的。该实现使用 C++ 预处理器和 Qt 所包括 的 moc（元对象编译器）。元对象编译器读取应用程序的头文件，并生成必要的代码，以支 持信号和槽机制。

qmake 生成的 Makefiles 将自动调用 moc，所有需要使用 moc 的编译规则都会给自动 的包含到 Makefile 文件中 。开发人员无需直接使用 moc 编辑、甚至无需查看生成的代码。

除了处理信号和槽以外，moc 还支持 Qt 的翻译机制、属性系统及其扩展的运行时类型 信息。比如，Q_PROPERTY()宏定义类的属性信息，而 Q_ENUMS()宏则定义在一个类中的枚举 类型列表。 Q_FLAGS()宏定义在一个类中的 flag 枚举类型列表，Q_CLASSINFO()宏则允许 你在一个类的 meta 信息中插入 name/value 对。它还使 C++ 程序进行运行时自检成为可能，并可在所有支持的平台上工作。

元对象编译器 moc（meta object compiler）对 C++文件中的类声明进行分析并产生用 于初始化元对象的 C++代码，元对象包含全部信号和槽的名字以及指向这些函数的指针。

moc 读 C++源文件，如果发现有 Q_OBJECT 宏声明的类，它就会生成另外一个 C++源文件，这个新生成的文件中包含有该类的元对象代码。例如，假设我们有一个头文件 mysignal.h，在这个文件中包含有信号或槽的声明，那么在编译之前 moc 工具就会根据该 文件自动生成一个名为 mysignal.moc.h 的 C++源文件并将其提交给编译器；类似地，对应 于 mysignal.cpp 文件 moc 工具将自动生成一个名为 mysignal.moc.cpp 文件提交给编译 器。

3.需要注意的问题

元对象代码是 signal/slot 机制运行所必须的。用 moc 产生的 C++源文件必须与类实现 文件一起进行编译和连接，或者用 #include 语句将其包含到类的源文件中。moc 并不扩展 #include 或者#define 宏定义,它只是简单的跳过所遇到的任何预处理指令。

13.4 Qt 的架构

Qt 的功能是建立在它所支持平台的底层 API 之上的，这使 得 Qt 非常灵活和高效。 Qt 使应用程序可与单平台的应用程序配套。

Qt 是一个跨平台的框架，它使用本地样式的 API 严格遵循每个支持平台中的用户界 面原则。Qt 绘制了 GUI 应用程序所需的几乎所有控件，并且开发人员可以通过重新实现虚 函数的方式来扩展或自定义所有这些控件。 Qt 的窗体能够精确模拟支持平台的观感，开发 人员还可以生成自己的自定义样式，为其应用程序提供具有鲜明特色的外观。

Qt 在它所支持的不同平台中使用底层 API。这与传统的“分层”跨平台工具套件不 同，传统工具套件是指在单个平台工具套件中使用的简单封装（例如，在 Windows 中使用 MFC；在 X11 中使用 Motif）。通常，分层工具套件速度较慢，其原因在于：库函数的每次 调用都会产生许多要经过不同 API 层的附加调用。分层工具套件往往会受到基本工具套件 的功能和行为的限制，导致应用程序中出现隐性错误。

Qt 做到了非常专业地支持各种平台 ，并且可以充分利用各种平台的优点。通过使用单 个源代码树，Qt 应用程序可以编译成每个目标平台的可执行程序。尽管 Qt 是一个跨平台 的框架，但与许多平台特定的工具套件相比， Qt 完全面向对象，更易于学习，更具有高效 性，这使得许多开发人员在开发单个平台时也更倾向于使用 Qt。

1.X11

Qt/X11 使用 Xlib 直接与 X 服务器通信。Qt 不使用 Xt（X Toolkit，即：X 工具套件）、Motif、Athena 或其他任何工具套件。

Qt 支持各种 Unix ：AIX®、FreeBSD®、HP-UX、Irix®、Linux、NetBSD、OpenBSD 和 Solaris。有关 Qt 所支持的编译器和操作系统版本的最新列表信息，请访问 NOKIA 公司网站。

Qt 应用程序自动适应用户的窗口管理器或桌面环境，并且在 Motif、CDE、GNOME 和 KDE 下具有桌面环境本身的观感。这与大多数 Unix 工具套件相反，这些套件总是把用户限 制在套件自身观感下。Qt 全面支持 Unicode。Qt 应用程序自动支持 Unicode 和非 Unicode 字体。Qt 将多种 X 字体组合在一起，可显示多语言文本。

Qt 的字体处理功能十分强大，可以在所有已安装的字体中搜索当前字体中不存在的字符。

Qt 可以充分利用 X 扩展程序。对于反锯齿字体、alpha 混合字体和矢量图形，Qt 支持 RENDER 扩展程序。Qt 还为 X 输入方法提供了现场编辑功能。Qt 可以 使用传统的多头 显示适配器和 Xinerama 支持多个屏幕。

Qt Application Source Code		
Qt API		
Qt/Windows	Qt/X11	Qt/Macintosh
GDI	X Windows	Carbon
Windows	Unix/Linux	Mac OS X

图 13-1 支持桌面平台中的 Qt 架构概览图

2.Microsoft Windows

Qt/Windows 使用 Win32® API 和 GDI 用于事件和绘图原语。Qt 不使用 MFC 或任何 其他工具套件。特别地，Qt 不使用缺乏灵活性的“常见”控件上，而是采用功能更强大的 可自定义的控件（如果不是特殊应用， Qt 使用 Windows 本身的文件和打印对话框） 。

使用 Windows 的客户可以在 Windows 98、NT4、ME、2000、XP 和 Vista 中使用 Microsoft Visual C++® 和 Borland C++来创建 Qt 应用程序。

Qt 为 Windows 版本执行运行检查， 并使用提供的最高级功能。 例如， 只有 WindowsNT4、2000、XP 和 Vista 支持旋转文本；Qt 则在所有 Windows 版本中都支持旋转文本，并在可能的情况下使用了操作系统本身的支持。 Qt 开发人员还可以避免处理不同版 本 Windows API 中的差异。

Qt 支持 Microsoft 的可访问界面。与 Windows 中的常见控件不同，您可以扩展 Qt 控件， 而不会丢失 Qt 基本控件的可访问或者说是固有信息。 另外，我们也可以制作和使 用自定义控件。Qt 支持 Microsoft Windows 下多个屏幕显示。

3.Mac OS X

Qt 将 Cocoa® 和 Carbon® API 组合在一起用来支持 Mac OS X。

Qt/Mac 引入了布局并直接支持国际化，允许采用标准化方式访问 OpenGL，并使用 Qt Designer 提供了功能强大的可视化设计。Qt 使用事件循环处理文件和异步套接字的输入输出。Qt 提供了稳定的数据库支持。开发人员可以使用流行的面向对象的 API 来创建 Macintosh 应用程序，该 API 具有综合文档和全部的源代码。

Macintosh 开发人员可以在自己喜欢的平台上创建应用程序，在其他受支持的平台 中，只需进行简单的重编译，即可显著扩大应用程序市场。 Qt 支持 Mac OS X 中通用的二 进制，这意味着可以为基于 Intel CPU 和 PowerPC CPU 的 Mac 创建 Qt 应用程序。

13.5 Qt 的事件模型

1.事件的概念

应用程序对象将系统消息接收为 Qt 事件。应用程序可以按照不同的粒度对事件加以 监控、过滤并做出响应。

在 Qt 中，事件是指从 QEvent 继承 的对象。Qt 将事件发送给每个 QObject 对象，这 样对象便可对事件做出响应。也就是说， Qt 的事件处理机制主要是基于 QEvent 类来实现 的，QEvent 类是其他事件类的基类。当一个事件产生时， Qt 就会构造一个 QEvent 子类的 实例来表述该事件，然后将该事件发送到相应的对象上进行处理。

编程人员可以对应用程序级别和对象级别中的事件进行监控和过滤。

2.事件的创建

大多数事件是由窗口系统生成的，它们负责向应用程序通知相关的用户操作，例如： 按键、鼠标单击或者重新调整窗口大小。也可以从编程角度来模拟这类事件。在 Qt 中大 约有 50 多种事件类型，最常见的事件类型是报告鼠标活动、按键、重绘请求以及窗口处理 操作。编程人员也可以添加自己的活动行为，类似于内建事件的事件类型。

通常，接收方如果只知道按键了或者松开鼠标按钮了，这是不够的。例如，它还必须 知道按的是哪个键，松开的是哪个鼠标按钮以及鼠标所在位置。每一 QEvent 子类均提供事 件类型的相关附加信息，因此每个事件处理器均可利用此信息采取相应处理。

3.事件的交付

Qt 通过调用虚函数 QObject::event() 来交付事件。出于方便起见， QObject::event()会将大多数常见的事件类型转发给专门的处理函数，例如： QWidget::mouseReleaseEvent()和 QWidget::keyPressEvent()。开发人员在编写自己的控 件时，或者对现有控件进行定制时，可以轻松地重新实现这些处理函数。

有些事件会立即发送，而另一些事件则需要排队等候，当控制权返回至 Qt 事件循环 时才会开始分发。Qt 使用排队来优化特定类型的事件。例如， Qt 会将多个 paint 事件压 缩成一个事件，以便达到最大速度。

通常，一个对象需要查看另一对象的事件，以便可以对事件做出响应或阻塞事件。这可以通过调用被监控对象的 QObject::installEventFilter() 函数来实现。实施监控对象 的 QObject::eventFilter() 虚函数会在受监控的对象在接收事件之前被调用。

另外，如果在应用程序的 QApplication 唯一实例中安装一个过滤器，则也可以过滤 应用程序的全部事件。系统先调用这类过滤器，然后再调用任何窗体特定的过滤器。开发人 员甚至还可以重新实现事件调度程序 QApplication::notify()，对整个事件交付过程进行 全面控制。

4.事件循环模型

Qt 的主事件循环能够从事件队列中获取本地窗口系统事件，然后判断事件类型，并将 事件分发给特定的接收对象 。 主 事 件 循 环 通 过 调 用 QCoreApplication::exec() 启 动 ， 随 着 QCoreApplication::exit()结束，本地的事件循环可用利用 QEventLoop 构建。作为事件分发器的 QAbstractEventDispatcher 管理着 Qt 的事件队列，事件分发器 从窗口系统或其他事件源接收事件，然后将他们发送给 QCoreApplication 或 QApplication 的实例进行处理或继续分发。QAbstractEventDispatcher 为事件分发提供了 良好的保护措施。

一般来说，事件是由触发当前的窗口系统产生的，但也可以通过使用 QCoreApplication::sendEvent()和 QCoreApplication::postEvent()来手工产生事件。需 要说明的是 QCoreApplication::sendEvent()会立即发送事件， QCoreApplication::postEvent()则会将事件放在事件队列中分发。如果需要在一个对象初 始化完成之际就开始处理某种事件，可以将事件通过 QCoreApplication::postEvent()发 送。

通过接收对象的 event()函数可以返回由接收对象的事件句柄返回的事件， 对于某些 特定类型的事件如鼠标（触笔）和键盘事件，如果接收对象不能处理，事件将会被传播到接 收对象的父对象。需要说明的是接收对象的 event()函数并不直接处理事件，而是根据被分 发过来的事件的类型调用相应的事件句柄进行处理。

5. 自定义事件

一般有下列 5 种方式可以用来处理和过滤事件，每种方式都有其使用条件和使用范围。

(1) 重载 paintEvent()、 mousePressEvent()等事件处理器（event handler）

重新实现像 mousePressEvent(), keyPressEvent()和 paintEvent()这样的 event handler 是目前处理 event 所采用的最常见的方法，这种方法比较容易掌握。

(2) 重载 QcoreApplication::notify()函数

这种方式能够对事件处理进行完全控制。也就是说，当你需要在事件处理器 (event handler) 之前得到所有事件的话，就可以采用这个方法，但是这样一来，因为只有一个 notify()函数，所以每次只能有一个子类被激活。这与事件过滤器不同，因为后者可以有任意数目并且同时存在。

(3) 在 QCoreApplication::instance()也即在 qApp 上安装事件过滤器

这样就可处理所有部件（widget）上的所有事件，这和重载 QCoreApplication::notify()函数的效果是类似的。 一旦一个 event filter 被注册到qApp(唯一的 QApplication 对象), 程序里发到其它对象的事件在发到其它的 event filter 之前,都要首先发到这个 eventFilter 上，不难看出，这个方法在调试（debugging）应用程序时也是非常有用的。

(4) 重载 QObject::event()函数

通过重新实现的 event()函数，我们可以在事件到达特定部件的事件过滤器（ event handler）前处理 Tab 事件。需要注意的是，当重新实现某个子类的 event()的时候,我们 需要调用基类的 event()来处理不准备显式处理的情况。

(5) 在选定对象（Object）上安装事件过滤器(event filter)

该对象需要继承自 QObject ,这样就可以处理除了 Tab 和 Shift-Tab 以外的所有事 件。当该对象用 installEventFilter()注册之后,所有发到该对象的事件都会先经过监测它 的 event filter。如果该 object 同时安装了多个 event filter,那么这些 filter 会按照 “后进先出”的规则依次被激活, 即顺序是从最后安装的开始，到第一个被安装的为止。

6.事件与信号的区别 需要注意，我们不应该混淆“事件”和“信号”这两个概念。

(1) 使用场合和时机不同 一般情况下，在“使用”窗口部件时，我们经常需要使用信号，并且会遵循信号与槽的机制；而在“实现”窗口部件时，我们就不得不考虑如何处理事件了。举个例子，当使用 QPushButton 时，我们对于它的 clicked()信号往往更为关注，而很少关心促成发射该信 号的底层的鼠标或者键盘事件。但是，如果要实现一个类似于 QPushButton 的类，我们就需要编写一定的处理鼠标和键盘事件的代码，而且在必要的时候，仍然需要发射和接收 clicked()信号。

(2) 使用的机制和原理不同

事件类似于 Windows 里的消息，它的发出者一般是窗口系统。相对信号和槽机制，它 比较“底层”，它同时支持异步和同步的通信机制，一个事件产生时将被放到事件队列 里，然后我们就可以继续执行该事件 “后面”的代码。事件的机制是非阻塞的。

信号和槽机制相对而言比较“高层”，它的发出者一般是对象。从本质上看，它类似 于传统的回调机制，是不支持异步调用的。

举个例子，在 QApplication 中有两个投送事件的方法：postEvent ()和 sendEvent()，它们分别对应 Windows 中的 PostMessage()和 SendMessage()，就是是异步 调用和同步调用， 一个等待处理完后返回，一个只发送而不管处理完与否就返回。

在应用中，涉及到底层通信时，往往使用事件的时候比较多，但有时也会用到信号和槽。

(3) 信号与槽在多线程时支持异步调用

在单线程应用时，你可以把信号与槽看成是一种对象间的同步通信机制，这是因为在 这种情况下，信号的释放过程是阻塞的，一定要等到槽函数返回后这个过程才结束，也就是 不支持异步调用。

从 Qt4 开始，信号和槽机制被扩展为可以支持跨线程的连接，通过这种改变，信号与 槽也可以支持异步调用了，这方面的内容涉及到多线程的很多知识，读者感兴趣的话，可以 参阅《C++ GUI Qt4 编程》中的相关内容。

13.6 构建 Qt 应用程序

利用一套工具，Qt 开发人员可以简化在所有支持平台中构建应用程序的流程。描述应 用程序、库和插件的项目文件被用来为每个平台生成适当的 makefile。

.pro 文件描述了各个项目，该文件以文本方式概述了源文件、头文件、 Qt Designer 窗体以及其他资源。这些资源都是由 qmake 工具来处理的，以便为每个平台中的项目生成 适当的 Makefile。

项目文件可描述 Qt 的所有库、工具以及示例。例如，只需以下三行即可描述 Qt 4 的 HTTP 示例：

```
HEADERS += httpwindow.h
SOURCES += httpwindow.cpp main.cpp
QT += network
```

前两个定义将构建此示例所需的头文件和源文件告知 qmake； 而最后一个定义则确保 使用 Qt 的网络连接库。使用项目文件语法，开发人员可以使用配置选项对编译流程进行精 细调节，并可为不同的部署环境编写各种有条件的编译规则。

此外，使用项目文件可以描述处于目录树层次较深位置的项目。 例如，Qt 示例位于顶 级 examples 目录内的目录树中。examples.pro 文件要求 qmake 深入到含有下列行的各类 示例的目录中：

```
TEMPLATE = subdirs
SUBDIRS = dialogs draganddrop itemviews layouts linguist \
mainwindows network painting richtext sql \
threads tools tutorial widgets xml
```

支持条件编译意味着 Windows 示例程序只有在 windows 操作系统下的 Qt windows 版 本时才会被编译。

```
win32:!contains(QT_EDITION, OpenSource|Console):SUBDIRS += activeqt
```

使用 qmake 编译项目时，编译套件中的其他工具自动会提供 Qt 的所有增强功能：moc将处理头文件以启用信号和槽；rcc 将编译指定的资源； 而 uic 则可用来根据用户界面创建代码。

pkg-config 集成支持预编译头文件， 可以生成 Visual Studio 项目文件；以及其他高 级功能，可以支持开发人员在针对常见项目组件使用跨平台构建系统的同时，还支持其利用 与平台特定相关的工具。

更为深入的信息，请参考本书的附录，那里有 qmake 以及 make 命令的详细介绍。 13.7 问题与解答

问：什么情况下可以断开信号与槽的关联？ 答：有 3 种情况需要断开信号与槽的关联： 1.断开与某个对象相关联的任何对象

这似乎有点不可理解，事实上，当我们在某个对象中定义了一个或者多个信号，这些 信号与另外若干个对象中的槽相关联，如果我们要切断这些关联的话，就可以利用这个方 法，非常之简洁。

```
disconnect( myObject, 0, 0, 0 );
```

或者

```
myObject-&gt;disconnect();
```

2.断开与某个特定信号的任何关联

```
disconnect( myObject, SIGNAL(mySignal()), 0, 0 )
```

或者

```
myObject-&gt;disconnect( SIGNAL(mySignal()) )
```

3.断开两个对象之间的关联。

```
disconnect( myObject, 0, myReceiver, 0 )
```

或者

```
myObject-&gt;disconnect( myReceiver )
```

在 disconnect 函数中 0 可以用作一个通配符，分别表示任何信号、任何接收对象、接 收对象中的任何槽函数。但是发射者 sender 不能为 0，其它三个参数的值可以等于 0。

问：Qt 的元对象系统还有哪些功能？

答: 除了提供在对象间进行通讯的机制外，元对象系统还包含以下几种功能：

1.QObject::metaObject() 方法

它获得与一个类相关联的 meta-object。

2.QMetaObject::className() 方法

在运行期间返回一个对象的类名，它不需要本地 C++编译器的 RTTI(run-time type information)支持。

3.QObject::inherits() 方法

它用来判断生成一个对象类是不是从一个特定的类继承出来，当然，这必须是在 QObject 类的直接或者间接派生类当中。

4.QObject::tr() and QObject::trUtf8()

这两个方法为软件的国际化翻译字符串 。

5.QObject::setProperty() and QObject::property()

这两个方法根据属性名动态的设置和获取属性值 。

除了以上这些功能外，它还使用 qobject_cast()方法在 QObject 类之间提供动态转 换，qobject_cast()方法的功能类似于标准 C++的 dynamic_cast()，但是 qobject_cast() 不需要 RTTI 的支持，在一个 QObject 类或者它的派生类中，我们可以不定义 Q_OBJECT 宏。如果我们在一个类中没有定义 Q_OBJECT 宏，那么在这里所提到的相应的功能在这个类 中也不能使用，从 meta-object 的观点来说，一个没有定义 Q_OBJECT 宏的类与它最接近的 那个祖先类是相同的，那就是说， QMetaObject::className() 方法所返回的名字并不是这 个类的名字，而是与它最接近的那个祖先类的名字。 所以，我们强烈建议，任何从 QObject 继承出来的类都定义 Q_OBJECT 宏。

问：在 Qt 中怎么响应事件，比如鼠标点击事件，键盘敲击事件等。

答：你需要在程序中重载下列函数，具体可以查阅 Qt Assistant。

```
mousePressEvent(QMouseEvent *event)
{
    //要做的事
}
keyPressEvent(QkeyEvent *event)
{
    //要做的事（键盘）
}
```

13.7 总结与提高

信号与槽作为核心机制在 Qt 编程中有着广泛的应用，相对于传统的对象间调用的通信 机制，它巧妙的避免了对象间的耦合，更适合于组件编程。本章首先介绍了信号与槽的基本 概念和用法、元对象系统以及在实际使用过程中应注意的一些问题，在本章的后半部分，介 绍了 Qt 的架构以及构建 Qt 程序的所必需的组件。

如何学习这些看起来有些枯燥的原理性的知识呢？ 作者的体会是,在初学的时候，不必钻研过于深入的原理和机制问题，在以后的学习实践中不断的结合体会，这时翻回头再来学习一些较为深入的知识，就容易的多。

附录 A qmake 使用指南

A.1 qmake 简介

简而言之，qmake 是用来编译 Qt 应用程序的。

编译工具的使用很大程度上的简化了 Qt 应用程序的编译工作。我们可以使用三种方法 来编译 Qt 应用程序：第一种方法是使用 Qt 提供的 qmake 工具，第二种方法是使用第三方 的编译工具，而第三种方法是使用集成开发环境（ IDE）。

qmake 工具可以使用与平台无关的.pro 文件生成与平台相关的 makefile。该工具包含 了调用 Qt 内置代码生成工具（moc、uic、和 rcc）的必要的逻辑规则。在本书的所有例子 中都使用了 qmake，在多数情况下会使用相对简单的.pro 文件。实际上，qmake 提供了很多 的特性，包括创建可以递归调用其他 makefile 的 makefile，以及根据目前平台切换相关特 性的开关状态等。

A.2 使用 qmake

qmake 工具是与 Qt 一起提供的。它用来编译 Qt 本身，并且生成 Qt 自带的工具和例 子。贯穿整书，我们一直使用 qmake 工程文件（.pro 文件）编译示例应用程序。本附录将 系统（但非全面）的学习.pro 文件的语法，并且会介绍几个 qmake 的基本概念。要想全面 了解它们，请参阅 qmake 指南的在线帮助文档。

A.2.1 .pro 文件语法

.pro 文件的目的是列举工程中包含的源文件。由于 qmake 用于编译 Qt 及其相关工具， 所以它很熟悉 Qt，并且能够生成一些触发 moc、uic、和 rcc 的规则。因此，qmake 的语法 很简明，而且很容易学习。

工具文件主要分为三种：app（单独的应用程序）、lib（静态和动态库）和 subfirs（递归编译）。工程文件的类型可以使用 TEMPLATE 变量指定如下：

```
TEMPLATE = lib
```

subdirs 模板可以用来编译子目录里的目标文件。在这种情况下，除 TEMPLATE = subdirs 外，还需要指定 SUBDIRS 变量。在每个子目录中，qmake 会搜寻以目录名命名 的.pro 文件，并且会编译该工程。如果没有 TEMPLATE 这一项，那么默认工程是 app。对于

app 或者 lib 工程，最常用使用的变量是下面这些：

- HEADERS 指定工程的 C++头文件（.h）。
- SOURCES 指定工程的C ++实现文件（.cpp）。
- FORMS 指定需要 uic 处理的由 Qt 设计师生成的.ui 文件。
- RESOURCES 指定需要 rcc 处理的.qrc 文件。
- DEFINES 指定预定义的 C++预处理符号。
- INCLUDEPATH 指定 C++编译器搜索全局头文件的路径。
- LIBS 指定工程要链接的库。库既可以通过绝对路径指定，也可以使用源自 Unix 的- L 和- l 标识符来指定（例如，-L/usr/local/lib 和-ldb_cxx）。
- CONFIG 指定各种用于工程配置和编译的参数。
- QT 指定所要使用的 Qt 模块（默认是 core gui，对应于 QtCore 和 QtGui 模块）。
- VERSION 指定目标库的版本号。
- TARGET 指定可执行文件或库的基本文件名，其中不包含任何的扩展、前缀或版本 号（默认的是当前的目录名）。
- DESTDIR 指定可执行文件放置的目录（默认值是平台相关的。例如，在 Linux 上，指当前目录；在 Windows 上，则是指 debug 或 release 子目 录）。
- DLLDESTDIR 指定目标库文件放置的目录（默认路径与 DESTDIR 相同）。

- CONFIG 变量用来控制编译过程中的各个方面。它支持下面这些参数：
    - debug 是指具有调试信息的可执行文件或者库，链接 Qt 库的调试版。
    - release 是指编译不具有调试信息的可执行文件或者库，链接发行版的 Qt 库。如果同时指定 debug 和 release，则 debug 有效。
    - warn_off 会关闭大量的警告。默认情况下，警告的状态是打开的。
    - qt 是指应用程序或者库使用 Qt。这一选项是默认包括的。
    - dll 是指动态编译库。
    - staticlib 是指静态编译库。
    - plugin 是指编译一个插件。插件总是动态库，因此这一参数暗含 dll 参数。
    - console 是指应用程序需要写控制台（使用 cout、cerr、qWarning()，等等）。
    - app_bundle 只适用于 Mac OS X 编译，是指可执行文件被放到束中，这是 Mac OS X 的默认情况。
    - lib_bundle 只适用于 Mac OS X 编译，指库被放到框架中。

要生成工程文件 hello.pro 的 makefile，可以输入：

```
qmake hello.pro
```

在这之后，可以调用 make 或 nmake 编译工程。通过键入以下命令，也可以使用 qmake 生成一个 Microsoft Visual Studio 工程（.dsp 或.vproj）文件：

```
qmake -tp vd hello.pro
```

在 Mac OS X 系统上，可以创建一个 Xcode 工程文件：

```
qmake -spec macx-xcode hello.pro
```

要创建 makefile，可以输入：

```
qmake -spec macx-g++ hello.pro
```

这里的-spec 命令行参数可以用来指定平台/编译器的组合。通常，qmake 可以正确的检测到所在的平台，但在某些情况下则由必要显式的指定平台的情况。例如，在 Linux 上 以 64 位模式调用 Intel C++编译器（ICC）生成 makefile，应当输入：

```
qmake -spec linux-icc-64 hello.pro
```

那些可用的规则在 Qt 的 mkspecs 目录中。

尽管 qmake 的主要用途是生成.pro 文件的 makefile，但也可以使用-project 参数在当 前目录下使用 qmake 生成.pro 文件，例如：

```
qmake -project
```

在这种模式下，qmake 将搜索当前目录下已知扩展名（.h、.cpp、.ui 等等）的文件，生成一个列举这些文件的.pro 文件。

本附录的余下部分将更详细的介绍.pro 文件的语法。一个.pro 文件中的条目的语法通 常具有如下形式：

```
variable = values
```

values 是字符串的列表。注释以井号（#）开头，在行尾处结束。例如：

```
CONFIG = qt release warn_off # I know what I'm doing
```

将列表["qt"，"release"，"warn_off"]赋给 CONFIG 变量，它会覆盖以前的各个值。

额外的操作符作为=操作符的补充。+=操作符可以用来扩展变量的值。因此：

```
CONFIG = qt
CONFIG += release
CONFIG += warn_off
```

这几行会和前面的例子一样，可以有效的把列表 ["qt"，"release"，"warn_off"] 赋值给 CONFIG 变量。-=操作符从当前的变量中移除所有出现的指定的值。因此：

```
CONFIG = qt realease warn_off
CONFIG -= qt
```

会使 CONFIG 的值变成["release"，"warn_off"]。*=操作在一个变量上添加一个值，但要求被添加的值不在变量的列表上；否则，就什么都不做。例如：

```
SOURCE *= main.cpp
```

这一行将把 main.cpp 实现文件添加到工程中，只有当还没有被添加的情况下才添加它。最后，=操作符使用指定的值替换符合正则表达式的值，这是一种类似于 sed（UNIX 流编辑器）的语法。

例如：

```
SOURCE ~= s/\.cpp\b/.cxx/
```

使用.cxx 替换 SOURCES 变量中所有.cpp 文件的扩展名。

在值的列表中，qmake 提供了访问其他 qmake 变量、环境变量和配置参数的方法。表附录 A-1 列举了这些语法。

表附录 A-1 qmake 变量语法

存取函数	说明
$$var/Name 或者$${varName}	.pro 文件中 qmake 变量在那一时刻的值
$$(varName)	当 qmake 运行时环境变量的值
$(varName)	当处理 makefile 时环境变量的值
$$[varName]	Qt 的配置参数值

A.2.2 qmake 的存取函数

前面的例子中使用的总是一些标准变量，例如 SOURCES 和 CONFIG，然而，我们也可以设置任意变量的值，并且可以使用 $$varName 或者$${varName}语法引用它。例如：

```
MY_VRESION = 1.2
SOURCE_BASIC = alphadialog.cpp \
main.cpp \
windowpanel.cpp
SOURCE_EXTRA = bezierextension.cpp \
xplot.cpp
SOURCES = $$SOURCES_BASIC \
$$SOURCES_EXTRA
TARGET = imgpro_$${MY_VERSION}
```

接下来的例子组合了前面介绍的几种语法，使用内置函数 $$lower()把字符串转换为小写：

```
# List of classes in the project
MY_CLASS = Annotation \
CityBlock \
CityScape \
CityView
# Append .cpp extension to lowercased class names,and add main.cpp
SOURCES = $$lower($$MY_CLASSES)
SOURCES ~= s/([a-z0-9_]+)/\1.cpp/
SOURCES += main.cpp
# Append .h extension to lowercased class names
HEADERS = $$lower($$MY_CLASSES)
HEADERS ~= s/([a-z0-9_]+)/\1.h/
```

有时可能需要在.pro 文件中指定包含空格的文件名。在这种情况下，只需简单的把文件名用引号括起来即可。

当在不同的平台上编译工程时，可能有必要基于平台指定不同的文件或者不同的参 数。qmake 的条件语法是：

```
condition
{
    then-case
}
else
{
    else-case
}
```

condition 部分可以是平台名字（例如，win32、unix、或者 macx），或者更复杂的断言。then-case 和 else-case 部分使用标准语法为变量赋值。例如：

```
win32
{
    SOURCES += serial_win.cpp
}
else
{
    SOURCES += serial_unix.cpp
}
```

else 分支是可选的。为了方便，当 then-case 部分仅有一条变量赋值，而且在没有 else-case 分支时，qmake 也支持单行形式的语法：

```
condition:then-case
```

例如：

```
macx:SOURCE += serial_mac.cpp
```

如果有几个工程文件需要共享相同的项，则可以把相同的项提取到单独的文件中，在各自的.pro 文件中使用 include()语句包含它们：

```
include(../common.pri)
HEADERS += window.h
SOURCES += main.cpp \
window.cpp
```

通常，打算被别的工程文件所包含的工程文件会带有 .pri（工程包含）的扩展名。

在前面的例子中，我们了解了$$lower()内置函数，它可以返回参数的小写版本。另外 一个有用的函数是$$system()，它允许我们从外部应用程序中产生字符串。例如，如果想要 确认当前使用的 UNIX 版本，可以这样写：

```
OS_VERSION = $$ system(uname -r)
```

然后，可以在条件中使用结果变量，并与 contains()合用：

```
contains(OS_VERSION,SunOS):SOURCES += mythread_sun.c
```

本附录只讲了一些表面的东西。qmake 工具提供了许多参数和特性，远多于这里所讲到的这些，包括对预编译头文件的支持、对 Mac OS X 通用二进制库的支持以及对用户定义的 编译器或者其他工具的支持等。对于这方面更多信息的了解，可以参阅 qmake 指南在线帮助文档。

附录 B make 命令

B.1 命令解释

make 命令本身可带有四种参数：标志、宏定义、描述文件名和目标文件名。其标准 形式为：

```
make [flags] [macro definitions] [targets]
```

Unix 系统下标志位 flags 选项及其含义为：

- -f file 指定 file 文件为描述文件，如果 file 参数为"-"符，那么描述文件指向标 准输入。如果没有"-f"参数，则系统将默认当前目录下名为 makefile 或者名为 Makefile 的文件为描述文件。在 Linux 中， GNU make 工具在当前工作目录中按照 GNUmakefile、 makefile、 Makefile 的顺序搜索 makefile 文件。
- -i 忽略命令执行返回的出错信息。
- -s 沉默模式，在执行之前不输出相应的命令行信息。
- -r 禁止使用 build-in 规则。
- -n 非执行模式，输出所有执行命令，但并不执行。
- -t 更新目标文件。
- -q make 操作将根据目标文件是否已经更新返回"0"或非"0"的状态信息。
- -p 输出所有宏定义和目标文件描述。
- -d Debug 模式，输出有关文件和检测时间的详细信息。

Linux 下 make 标志位的常用选项与 Unix 系统中稍有不同，下面我们只列出了不同部 分：

- -c dir 在读取 makefile 之前改变到指定的目录 dir。
- -I dir 当包含其他 makefile 文件时，利用该选项指定搜索目录。
- -h help 文挡，显示所有的 make 选项。
- -w 在处理 makefile 之前和之后，都显示工作目录。

B.2 使用 make 自动构建

make 实用程序是一个工具，用于控制构建以及重构软件的过程。make 将构建什么软 件、如何构建以及何时构建这些过程自动化了，使程序员能够专注于编写代码。因为它包 含的逻辑可调用适于 GCC 编译器的选项和参数，所以节省了很多输入操作。另外，它还可 以帮助您在构建应用程序时，不会在输入所有复杂命令时出现错误；相反，只需输入一个 或两个 make 命令即可。通过本节的学习可以熟悉 makefile 的外观和布局。

除去最简单的软件项目外，对于其他软件项目而言 make 是根本。首先，由多个源文 件组成的项目要求长且复杂的编译器调用。make 简化了这项工作，方法是在 makefile 中存储这些难记的命令，makefile 是一个文本文件，包含构建该软件项目所需的所有命 令。

make 对想要构建程序的开发者和用户而言都很方便。当开发者修改某个程序后，无 论是增加新功能还是合并 bug 修复，make 允许使用单条短的命令即可重构该程序。make 对用户也是很方便的，因为他们不必阅读大量详细解释如何构建程序的文档。相反，他们 只需输入 make，然后是 make test，最后是 make install。多数用户都喜欢这种简单构 建指令的便利。

最后，make 加快了编辑一编译一调试过程。它最大限度的降低了重构时间，因为它 的智能可以确定哪个文件被修改了，并且只重新编译被修改过的文件。

附录 C Qt 资源

C.1Qt 官方资源

全球各大公司以及独立开发人员每天都在加入 Qt 的开发社区。他们已经认识到了 Qt 的架构本身便可加快应用程序开发进度。这些开发人员，无论是想开发单平台软件、还是想 开发跨平台软件，都可从 Qt 统一而直接的 API、强大的构建系统以及各种支持工具（例如 Qt Designer）中受益无穷。

Qt 具有一个极具活力并十分有益的用户社区，用户可以通过以下方式进行沟通： qt- interest 邮件列表、Qt Centre 网站（网址为：www.qtcentre.org）以及其他社区网站和 博客。另外，许多 Qt 开发人员也是 KDE 社区的活跃成员。Qt 客户每个季度都会收到我们 的开发人员新闻通讯《Qt Quarterly》。如今，Qt 的用户社区还提供了越来越多的第三方 商业软件和开源软件；有关最新信息，请访问 www.qtsoftware.com。

Qt 的一系列文档可在线访问，网址为：doc.qtsoftware.com。另外，有关 Qt 编程 的详细介绍，市场上还有一系列英语、法语、德语、俄语、日语以及中文版本书籍。 Qt 的 官方书籍是《C++ GUI Programming with Qt 4》(ISBN 0-13-187249-4)。

Nokia 公司及其合作伙伴为 Qt、Qtopia 和 C++ 提供了一系列培训选择，包括针对大 众客户提供的开放式课程培训以及针对具有特殊需求的客户所提供的现场培训。有关详细信 息，请访问 www.qtsoftware.com/training/。

除了为 C++ 开发人员提供综合框架以外，Qt 还可以使用其他语言编程。Qt 本身包含 了 QtScript 模块，这是一种类似 JavaScript 的技术，使用这一技术，开发人员可以用编 写脚本的方法让用户访问应用程序的某些特定区域。

Nokia 公司还提供了 Qt Jambi （现已交由社区开发）这一技术，使用这种技术， Java 开发人员可以基于 Java 编程语言来使用 Qt。

另外，Nokia 公司及第三方公司针对 JavaScript、Python、Perl 和 Ruby 提供了语言 绑定；其中许多解决方案都是由开源开发团队提供并加以维护的。

开发人员可在自己喜欢的平台上免费试用 Qt，时间期限为 30 天，并可得到 Qt Software 的服务和支持。有关详细信息，请发送电子邮件至 info@qtsoftware .com。

C.2 Qt 开发社区

C.2.1 国际社区

下面是在线参考的一些国际网址：

- http://partners.trolltech.com/
- http://lists.trolltech.com/qt-interest/
- http://doc.trolltech.com/qq/

可以从 http://doc.trolltech.com 中获取 Qt 的当前版和一些早期版本的在线参考文 档。这个网站也选摘了 Qt 季刊（Qt Quarterly）中的一些文章。Qt 季刊是 Qt 程序员的时 事通讯，会发送给所有获得 Qt 商业许可协议的人员。

在使用 Qt 时，如果有问题需要进行交流讨论，可以访问一些 Qt 相关的技术论坛。国 外较好的论坛有：

- http://www.qtform.org
- http://www.qtcentre.org
- http://www.qt-apps.org

建议大家经常去浏览一下，会有很多的收获，但是技术文章往往有一定的深度，初学者可能会不太适应。

C.2.2 国内社区

国内几个关于 Qt 的论坛如下：

- Qt 中文论坛：http://www.qtcn.org

  是国内最为老牌和最为活跃的 Qt 中文社区，内容丰富、覆盖面广、在线人数众多，上 面有很多热心、无私的 Qt 爱好者，他们会帮助初学者尽快的入门。

- Qt 知识库：http://www.qtkbase.com

  定位类似于 VC 知识库的技术百科全书类型的网站，有很多的原创技术文章，具有相当的参考价值。

- Qt 核心技术网：http://www.insideqt.com

  具有鲜明技术特色的网站，主要关注 Qt 技术的内涵和应用，其中《Inside Qt》系列 文章对 Qt 的技术实现内幕作了一定深度的探讨，广受 Qt 开发者的欢迎。

- 酷享 Qt 论坛：http://www.cuteqt.com/bbs

网站的名称来自于 Qt 的正确英文发音，堪称国内最有原创精神的 Qt 综合技术网站， 其管理团队由 Qt 中文界的好手和爱好者组成。网站内容涉及 Qt 技术的方方面面，并且具有相当的深度和实用价值。

- Qt Everywhere 网站：http://www.qteverywhere.com

  由来自 Nokia 的 Zhang Chi 创建，侧重 Qt 最新的新闻、事件、活动、培训以及编程等等相关的内容，内容精彩并且富有新意，更新比较及时，也是笔者经常访问的网站之一。

  需要特别指出的是其 Qt 学习子版块，网址是 http://www.qteverywhere.com/learnqt，里面 有许多深入浅出的 Qt 学习心得文章，资料，教程，可以帮助大家由浅入深，从入门到精通 的学习 Qt。这里也向 Qt 的读者朋友推荐。

- CSDN 网站之 Qt 技术社区：qt.csdn.net

  CSDN 也在持续的关注 Qt 的最新进展，Qt 技术社区正也已经正式在 CSDN 上线了，大家 可以在 CSDN 的首页点击链接进入，也可以直接访问 qt.csdn.net。

- 还有就是在 IBM 的 DevelopWorks 网站上有关 Qt 的网页，其中有国内早期号称 Linux 三剑客之一的于明俭的一些文章，时至今日，仍然有参考的价值。

C.2.3 有关 Qt 的博客 专门关注 Qt 的博客有：

- 酷享 Qt 论坛之博客：http://www.cuteqt.com

  酷享 Qt 论坛的博客汇聚了国内的 Qt 好手坐镇，行文流畅，言简意赅，富有探究精 神，与 Qt Everywhere 论坛的博客内容互为补充。

- Qt Everywhwere 论坛之博客：http://www.qteverywhere.com

  是目前国内 Qt 中文论坛中访问量最大、关注度最高的博客，内容翔实，图文并茂，与 酷享 Qt 论坛的博客内容互为补充。

- 齐亮的个人主页：http://www.qiliang.net

  齐亮目前就职于 Qt Software，是《C++ GUI Qt3 编程》一书的中文译者，他目前生活工作在挪威的奥斯陆。他的博客内容丰富，在讨论 Qt 技术的同时富有人文色彩。

- Qt 中文论坛（http://www.qtcn.org）、Qt 知识库网站等也有自己的博客，不过更新速度不是很快。

其他还有 Qt 界一些同仁的博客，都各具特色，推荐大家访问。

- leenux 的 Qt 博客： http://leenux.cublog.cn/
- pplinux 的 Qt 博客： http://blog.chinaunix.net

- 昆仑的 blog：http://www.cnblogs.com/ttylikl/